Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Janez Barbič

Neposredna prodaja pohištva preko mobilne aplikacije

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: dr. Andrej Brodnik

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Pohištvena industrija je glede na velike zaloge lesa v Sloveniji ena strateško najključnejših industrij. Žal pa ne dosega visoke stopnje dodane vrednosti, ker običajno ne nagovarja potrošnika neposredno. To nišo zapolnjujejo mizarske delavnice, ki so edine sposobne vzpostaviti neposredni prodajni kanal do kupca.

Razvijete spletno/mobilno aplikacijo, ki bo vzpostavila neposredni prodajni kanal med pohištveno industrijo in končnim kupcem. Preko kanala naj ima kupec možnost neposredno nagovoriti arhitekta v pohištvenem podjetju ter z njim sooblikovati končni izdelek. S tem, osebni stikom, bo kupec tudi dobil občutek višje varnosti in kakovosti izdelka.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Janez Barbič, z vpisno številko **63030132**, sem avtor diplomskega dela z naslovom:

Neposredna prodaja pohištva preko mobilne aplikacije

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom dr. Andreja Brodnika,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 29. septembra 2014 Podpis avtorja:

Družini, mentorju in njegovim asistentom ter študijskim kolegom - hvala.

Svoji dragi Irmi.

Slike

2.1 Arhitektura operacijskega sistema Android. 15

3.1 Tipični pogled razvojnega okolja Eclipse, s pomočjo katerega je bila razvita spletna storitev. 24
3.2 Tipični pogled orodja za razvoj aplikacij Android. 26

4.1 Arhitektura sistema. 28
4.2 Podatki o razširjenosti mobilnih operacijskih sistemov v Sloveniji od junija 2008 do novebra 2013 [8]. 29
4.3 Delež verzij operacijskega sistema Android na mobilnih napravah Android [2]. 30
4.4 Diagram aktivnosti aplikacije Android in njenih stanja [12]. . . 32
4.5 Shema aktivnosti aplikacije. 36
4.6 Shema storitve mobilne aplikacije, ki teče v ozadju. 45
4.7 Shema spletne storitve ki streže zahtevkom, ki pridejo do strežnika. 50
4.8 Struktura podatkovne zbirke 54

5.1 Pogled testnega zaslona na napravi ASUS Nexus 7 2012. . . . 56
5.2 Pogled testnega zaslona na napravi SAMSUNG Galaxy Tab 4. 57
5.3 Pogled testnega zaslona na napravi SONY Xperia M. Prikazan je izbirni meni. 58
5.4 Pogled testnega zaslona na napravi SONY Xperia M. Prikazana je izbera elementa menija. 58

SLIKE

Kodni izseki

4.1 Osnovni atributi abstraktnega razreda DataObject 38
4.2 Konstruktor razreda DataObject. 39
4.3 Dostop do atributa state. 39
4.4 Abstraktne metode razreda DataObject 39
4.5 Implementirana metoda store(int id) v razredu QuestionnaireObject. 40
4.6 Metoda getURL ki vrne lokacijo vira. 41
4.7 Funkcija ki strežniku pošlje izpolnjen vprašalnik v obliki objekta JSON. 42
4.8 Zagon storitve mobilne aplikacije v ozadju. 46
4.9 Implementacija niti s pomočjo katere storitev mobilne aplikacije osvežuje podatke. 46
4.10 Primer dela zgradbe spletne storitve. 51

Kazalo

Povzetek

Abstract

1 Uvod **1**

1.1 Predstavitev problema 1

1.1.1 Izhodišče 1

1.1.2 Potrebe 2

1.1.3 Predlagane rešitve 3

1.2 Diplomsko delo kot rešitev podproblema 4

1.3 Predstavitev obstoječih rešitev 5

2 Uporabljene tehnologije pri razvoju **9**

2.1 Spletni strežnik Apache Tomcat 9

2.2 Spletna storitev 10

2.3 MySQL in SQL 11

2.4 Mobilna platforma Android 13

2.4.1 Arhitektura operacijskega sistema Android 14

2.5 Arhitektura REST 18

2.5.1 Omejitve 19

2.5.2 Kako uporabiti arhitekturo REST pri snovanju spletnih storitev 21

3 Razvojna orodja **23**
3.1 Eclipse 23
3.2 Razvojno okolje Android 25

4 Razvoj sistema EkoBuMDesign **27**
4.1 Arhitektura sistema 27
4.2 Mobilna aplikacija kot odjemalec 29
4.2.1 Arhitektura aktivnosti 30
4.2.2 Podatkovna zbirka SQLite 34
4.2.3 Aktivnosti mobilne aplikacije 35
4.2.4 Splošne nastavitve 37
4.2.5 Stanje aplikacije 37
4.2.6 Vmesnik podatkovne zbirke 41
4.2.7 Razred REST 41
4.2.8 Osnovni tok dogodkov aktivnosti mobilne aplikacije . . 43
4.2.9 Storitev mobilne aplikacije v ozadju 44
4.2.10 Povezovalna storitev 44
4.2.11 Osnovni tok dogodkov storitve mobilne aplikacije . . . 47
4.3 Spletna storitev 49
4.3.1 Arhitektura spletne storitve 49
4.4 Podatkovna zbirka 53

5 Testiranje **55**
5.1 Testirani sistemi 55
5.2 Poraba sistemskih virov 57
5.3 Spletna storitev 59

6 Sklepne ugotovitve in nadaljnje delo **61**

Povzetek

Namen diplomske naloge je izdelava sistema za naročanje pohištva preko mobilne aplikacije in izdelava prototipa mobilne aplikacije na platformi Android. Rezultat praktičnega dela je prototip spletne storitve in mobilne aplikacije. Aplikacija je izdelana za naprave z operacijskim sistemom Android.

Izdelek omogoča stranki naročanje pohištva s pomočjo mobilne aplikacije. Naročnik prostor fotografira in izpolni vprašalnik o namembnosti prostora. Povratna informacija vsebuje vizualizacijo, ki jo izdela arhitekt podjetja. Naročnik si lahko vizualizacijo interaktivno ogleda v prostoru, ki je predmet naročila. Poleg tega mobilna aplikacija ponuja še nekaj standardnih lastnosti kot so osebne nastavitve, opozorila ob prejetju sporočila idr.

Ključne besede: mobilne tehnologije, Android, Android aplikacija, EkoBUmDesign.

Abstract

The goal of the thesis is to produce a system for ordering furniture through mobile applications on the Android platform. The result of practical work is a prototype of a web service and mobile application. The application is designed for any Android powered device.

The Android application allows customers to order furniture with the help of mobile application. The customer takes pictures of a room and completes a questionnaire about the room and sends both to the company. Company architect's feedback includes an interactive visualisation of the furnished room drawn by the company's architect. The client can then view the visualisation of the room in question as part of the order. In addition the mobile application offers some add-on features such as personal preferences and alerts upon receiving messages.

Keywords: mobile technologies, Android, Android application, EkoBUm-Design.

Poglavje 1

Uvod

Današnji način življenja se zaradi vse hitrejše globalizacije močno razlikuje od načina življenja že samo deset let nazaj. Hiter razvoj in razširjena uporaba mobilnih tehnologij nam omogoča še hitrejši in bolj neposreden dostop do storitev. Predmet tega diplomskega dela je ravno rešitev, ki temelji na mobilnih tehnologijah in neposredno povezavo ponudnika in odjemalca storitve brez vmesnih členov.

1.1 Predstavitev problema

1.1.1 Izhodišče

Slovenska pohištvena industrija je ob svetovni ekonomski krizi prišla do točke, ko se bo morala soočiti s svojo nekonkurenčnostjo. Podjetje Brest je v odgovor na ta problem izvedlo spremembo poslovnega modela. Krepitev regionalnega razvojnega potenciala naj bi pripeljala do večje inovativnosti, novih izdelkov, procesov in materialov in novih produktov v višjih cenovnih razredih.

V nakupni proces vsakodnevnih izdelkov se običajno vlaga malo napora, ko pa gre za nakup izdelkov z višjo ceno in izdelke za dolgotrajnejšo uporabo, pa se kupec navadno bolj angažira, saj zaznava višje finančno tveganje. Kupec vloži več energije, da pridobi čimveč informacij o pohištvu in je z izbiro

zadovoljen.

Podjetja, ki se zavedajo prednosti, ki jih prinaša veliko število stalnih kupcev, se morajo novo nastalim razmeram učinkovito prilagajati. Raziskave kažejo, da imajo moderni kupci vse več zahtev, življenjska doba pohištva pa je vse krajša. Podjetja morajo iz tega razloga natančno spremljati, kakšne so želje kupcev in jih tudi zadovoljiti. Zaradi dinamičnega načina življenja modernega kupca je potrebno strankam nuditi dodatne možnosti pri nakupovanju izdelkov kot so nakupi ob koncu tedna, nakupi prek interneta, dostava ob želenem času, kvalitetne prodajne in poprodajne storitve in drugo. Podjetja, ki se tega ne zavedajo, bodo pričela zaostajati za konkurenco.

Osebna obavnava je ena izmed zaželjenih storitev kupca. Nekatere panoge to storitev kupcu že ponujajo, kot je na primer avtomobilska industrija, kjer je že pred izdelavo avtomobila znan končni kupec in je avtomobil delan po željah kupca.

Raziskave mobilnega trga kažejo, da vse več globalne populacije vseh starostnih segmentov uporablja vsaj eno mobilno napravo. Tudi vse več spletnih nakupov se opravi ravno s pomočjo mobilnih naprav. Z mobilnimi aplikacijami je mogoče doseči nov segment kupcev še na nov način. Možnost so med drugim ponudba prodajnega kataloga, naročilo storitve, pa tudi ponudba iger, ki so vezane na produkte. Multimedijske vsebine, ki jih potencialni kupci generirajo s pomočjo iger, pa si kupci lahko delijo na socialnih omrežij.

Prodor mobilnih tehnologij tako nakazuje, da bo tudi nakupovanje pohištva postala mobilna storitev.

1.1.2 Potrebe

Potrošnikom podjetje Brest želi predstaviti in ponuditi izdelke, ki ustvarjajo kvaliteto bivanja, jih je mogoče razgraditi in reciklirati. Poleg tega jim podjetje želi predstaviti svoje aktivnosti, s katerimi skrbi za njihovo zadovoljstvo od vgradnje do razgradnje in recikliranja pohištva, to je v celotnem življenjskem ciklu izdelka.

EU svojo pohištveno in lesno industrijo ščiti s tako imenovanimi kva-

litativnimi ukrepi, kamor prištevamo okoljsko in zdravstveno regulativo za izdelke. Že z vidika same zaščite, še bolj pa z vidika vse večje okoljske in zdravstvene osveščenosti prebivalstva Evrope lahko pričakujemo v naslednjih petih letih popolno uveljavitev koncepta *cradle to cradle*[1] pri lesenem pohištvu in drugih lesenih izdelkih. Tovrstna uveljavitev bo pripeljala do intenzivne zamenjave starega – z okoljem in zdravjem neskladnega – pohištva z novimi bivanjskimi rešitvami pri opremi in obenem visoko potrebo po recikliranju starega pohištva in predelavi le-tega v celoten spekter izdelkov, surovin in materialov. V Brestu se takšnega poteka zavedajo in želijo sooblikovati tehnološke postopke sodobne razgradnje, prav tako pa tudi vzpostaviti celoten sistem zbiranja in predelave starega pohištva ter vračanje predelanih komponent nazaj v uporabo. Sicer pa je dimenzija problema prevelika za podjetje te velikosti in bodo pri iskanju rešitev vključili tudi druge proizvajalce pohištva, predvsem pa sodelavce iz inštitucij znanja in raziskav.

1.1.3 Predlagane rešitve

Cilj predlaganega projekta je razviti mobilno storitev in zanjo aplikacijo, s katero bo podjetje potrošniku olajšalo nakupno odločanje pri nakupu pohištva. Potrošnik bo preko aplikacije in zalednega sistema neposredno komuniciral z arhitektom (načrtovalcem), s katerim bosta skupaj ustvarila pohištvo po meri zanj in za njegovo stanovanje. Aplikacija bo omogočala:

1. preprost zajem potrebnih podatkov za načrtovanje pohištva po meri in
2. neposredno komunikacijo z arhitektom - proizvajalčevim načrtovalcem, ki bo lahko na podlagi zajetih podatkov načrtal pohištvo po meri.

[1]Cilj koncepta *od zibelke do zibelke (ang. cradle to cradle)* je izdelati postopke, ki niso le učinkoviti, ampak tudi ne proizvedejo odpadkov. Vsi vhodni in izhodni postopki in surovine so tretirani kot biološka ali tehnična hranila. Tehnična hranila se lahko ponovno uporabijo ali reciklirajo brez izgube kvalitete, biološka pa se lahko kompostirajo ali iztrošijo. V nasprotju pa koncept *od zibke do groba (ang. cradle to grave)* poudarja odgovornost podjetja za odstranitev produkta, ki ga je izdelalo, vendar ni nujno, da komponente produkta vrne v uporabo.

Za zajem dejavnikov bodo uporabljene zmožnosti sodobnih mobilnih naprav, ki vključujejo uporabo visoko razvitih uporabniških vmesnikov vključno z rabo vgrajenih senzorjev (slika ipd.). Razvita mobilna aplikacija se bo osredotočila na vse stopnje procesa odločanja pri nakupu pohištva, od prepoznavanja potreb, iskanja informacij, ocenjevanje možnosti in nakupno odločitev do ponakupnega vedenja.

Aplikacija bo razvita v dveh sklopih. Prvi sklop bo predstavljala spletna e-storitev, ki bo ponujena v oblaku, kar pomeni samodejno povečljivost in zanesljivost. Storitev bo vključevala sodelovanje z zalednimi sistemi pri načrtovalcih in ne nazadnje v sami proizvodnji. Drugi sklop bo predstavljala mobilna aplikacija, ki bo uporabljala omenjeno aplikacijo. Mobilna aplikacija bo razvita za mobilne naprave z operacijskim sistemom Android.

Predlagani projekt je zgolj prvi korak na poti k celoviti osebnostni obravnavi potrošnika. Tako bo na podlagi zbranih podatkov moč izmenjevati informacije o ponakupnem vedenju ter ponuditi kupcu višjo mero občutka varnosti, saj bo proizvajalec skrbel za njegovo pohištvo takorekoč celoten življenjski cikel. Ne nazadnje bo ponudnik pohištva lahko pridobil povratno informacijo o ponakupni izkušnji kupca, njegovo zadovoljstvo ali nezadovoljstvo z izdelkom, kar vpliva na nadalnje nakupe kupca in strategije razvoja ponudnika. Poudariti velja še to, da obstoj življenjskega cikla takoj pomeni, da lahko ponudnik kupcu občasno ponudi zamenjavo ali nadgradnjo obstoječega izdelka ali celo odloženi nakup.

1.2 Diplomsko delo kot rešitev podproblema

Rešitev celotnega problema presega okvirje diplomske naloge, zato se to delo ukvarja z delom problema, kako iz prodajnega procesa izločiti posrednika in hkrati izboljšati uporabniško izkušnjo že od odločanja stranke o nakupu. Zastavljeni cilji pred začetkom izdelave diplomske naloge so bili:

- definiranje komnikacijske poti arhitekt - stranka s pomočjo mobilnih naprav. Naloga vključuje iskanje ustreznih tehnologij, ki so potrebne za

realizacijo komunikacijske poti in implementacijo, nastavitev oziroma uporabo le teh,

- izdelava prototipa mobilne aplikacije ki omogoča stranki naročilo storitve podjetja. Prototip naj omogoča, da s pomočjo mobilne aplikacije podjetje (oziroma arhitekt) od stranke pridobi vse potrebne informacije za oblikovanje notranjosti prostora. Nato naj vključuje še možnost prejetja odgovora arhitekta. Nižje na prioritetni lestvici je prejemanje ostalih informacij kot so informacije o akcijski ponudbi, poprodajnih storitvah idr.,
- izdelava strežniške aplikacije, ki bo komunicirala z mobilno aplikacijo. Vključuje naj podatkovno zbirko, kjer se hranijo podatki o strankah, naročilih in ostalih podatkih, ki se bodo tekom razvoja izkazali za potrebne.

Zastavljeni cilji služijo kot kriteriji za ovrednotenje končnega izdelka.

1.3 Predstavitev obstoječih rešitev

Pogoj, da ponujeno rešitev trg sprejme, je poznavanje konkurence. Ker se izdelana aplikacija ukvarja z opremljanjem prostora prodajo pohištva so v raziskavo zajete aplikacije, ki se ukvarjajo s podobno tematiko. Posebno pozornost velja posvetiti naboru funkcionalnosti, uporabniškemu vmesniku in razlogom, zakaj uporabniki aplikacijo prenesejo na svojo napravo, jo uporabljajo in ji na distribucijskem servisu prisodijo visoko oceno. Potrebno je tudi preudariti, katere funkcionalnosti konkurence lahko v enaki ali boljši meri implementira rešitev problema, s katerim se ukvarja diplomska naloga.

Na področju oblikovanja interierjev obstajata dve aplikaciji, ki sta v številu prenosov močno v prednosti glede na konkurenčne izdelke, ponujata pa tudi veliko več različnih možnosti poleg osnovne, ki jo ponujajo tudi ostale aplikacije s tega področja, to je predstavitev oblikovalskih idej uporabniku.

Houzz Interior Design Ideas

Aplikacija ponuja [5]:

- bazo fotografij prostorov, kjer uporabniki lahko poiščejo navdih za opremljanje ali dekoracijo prostorov. Fotografije so razdeljene po kategorijah po namembnosti prostora,
- deljenje mnenj z uporabniki aplikacije v sami aplikaciji,
- deljenje mnenj, slik in idej s pomočjo socialnega omrežja Facebook,
- bližnjice do nakupa dizajnerskih kosov pohištva, dekoracijskih predmetov, svetil itd., ki so del oblikovne celote s fotografije,
- nabor ponudnikov stanovanjske opreme in storitev glede na uporabnikovo trenutno lokacijo,
- nasvete kako popolnoma preobraziti podobo prostora, na primer s pleskarskimi nasveti.

Uporabnik se avtenticira s pomočjo Facebook računa ali razvijalčevega lastnega sistema. Prijava je potrebna za uporabo vseh funkcionalnosti.
Število prenosov aplikacije: 1.000.000+
Ocena: 4.8/5.0

Homestyler Interior Design

Aplikacija ponuja [4]:

- izdelavo lastnega dizajna prostora. Prostor uporabnik najprej fotografira tako, da v objektiv ujame poljuben vertikalen kot prostora. V fotografijo umesti kvader tako, da s kotom prostora poravna vertikalni rob kvadra, nato poravna horizontalne robove kvadra z robovi prostora. Nato določi višino vertikalnega roba kvadra in scena je pripravljena na oblikovanje vendar prostora ni moč obračati, orientacija scene je fiksna,
- opremljanje scene s prednaloženimi 3D modeli pohištva in ostale opreme,
- 3D modele pohištva, ki so izdelani po izdelkih izdelovalcev pohištva in oblikovalskih predmetih ter jih je mogoče tudi kupiti,
- ogled izdelanih scen ostalih uporabnikov in tudi poklicnih oblikovalcev,

- deljenje izdelanih scen z uporabniki socialnega omrežja Facebook,
- iskanje poklicnih oblikovalcev v uporabnikovi bližini skupaj s kontaktnimi informacijami,
- pretakanje idejnih fotografij ki ne ponazarjajo nujno celotnih interierjev, lahko predstavljajo le posamezen oblikovalski predmet,
- oblikovalsko revijo, ki vsebuje članke o modnih barvnih kombinacijah, najnovejših oblikovalskih dosežkih s področja opremljanja interierjev, nasvete pri redekoraciji prostorov in podobno.

Uporabnik se avtenticira s pomočjo Facebook računa ali razvijalčevega lastnega sistema. Prijava je potrebna za uporabo vseh funkcionalnosti.
Število prenosov aplikacije: 1.000.000+
Ocena: 4.2/5.0

Serija aplikacij Decoration Designs

To je serija aplikacij, ki so si v samem bistvu enake, razlikujejo se le po namembnosti prostora, za katerega vsebujejo idejne fotografije. Aplikacije vsebujejo le fotografije opremljenih prostorov, ki uporabniku služijo kot navdih pri opremljanju. Omogočajo deljenje fotografij preko Bluetooth povezave.
Število prenosov aplikacije: okrog 10.000+
Ocena: 3.7/5.0
Ocena nekoliko variira glede na pozamezno aplikacijo iz serije.

Nekatere večje trgovske verige, kot je na primer Ikea, ponujajo mobilne aplikacije, ki so dejansko le interaktivni katalogi pohištva. Ker je načrtovana rešitev veliko več kot le katalog, takšnih alikacij nismo preizkušali.

Najpomembnejša prednost aplikacije EkoBUmDesign je omogočiti uporabniku izdelavo interierja poklicnega oblikovalca - arhitekta. Naslednja prednost je ogled celotnega prostora v treh dimenzijah. Največ, kar omogočajo najbolj izpopolnjeni izdelki, je pogled na prostor iz stacionarnega kota. Podoben sistem katerega koli slovenskega podjetja ne obstaja, torej je sistem EkoBUmDesign pionirski projekt za slovenski trg. Ker pa bo aplikacija do-

stopna globalno je dobro načrtovati aplikacijo z možnostjo razširitve na tuje trge.

Poglavje 2

Uporabljene tehnologije pri razvoju

Večina sistemov, ki se pojavljajo v računalništvu, sledi modelu odjemalec-strežnik. Če želimo takšen sistem izdelati, imamo na voljo že obstoječe podsisteme in tehnologije, ki omogočajo navedeni način delovanja. Na voljo je tudi več enakovrednih sistemov, ki so lahko plačljivi, lahko pa so tudi brezplačni. Za rešitev, s katero se ukvarja diplomska naloga, so izbrani odprtokodni sistemi.

S strojno opremo se ta diplomska naloga ne ukvarja. Predpostavi se, da le ta zadostuje pogojem za delovanje sistema.

2.1 Spletni strežnik Apache Tomcat

Apache Tomcat je odprtokodni spletni strežnik in vsebovalnik servletov, ki ga razvija in vzdržuje neprofitna organizacija Apache Software Foundation (ASF). Tomcat implementira specifikaciji Java Servlet in JavaServer Pages (JSP) podjetja Sun Microsystems in ponuja čisto Javansko HTTP spletno strežniško okolje, v katerem teče javanska koda. Konfiguriranje in upravljanje strežnika je mogoče s pomočjo priloženih orodij, lahko pa se do vseh funkcionalnosti dostopa v ustreznih XML datotekah [16]. Tomcat se lahko

uporablja kot samostojen produkt z lastnim spletnim strežnikom ali pa skupaj z ostalimi spletnimi strežniki kot so Apache, Netscape Enterprise Server, Microsoft Internet Information Server (IIS) in Microsoft Personal Web Server. Tomcat potrebuje JRE 1.1 (izvajalno okolje Java) ali novejše.

2.2 Spletna storitev

Spletna storitev *(ang. Web service)* je računski sistem, ki si izmenjuje XML sporočila z drugimi sistemi s pomočjo HTTP komunikacijskega protokola. Poleg tega je spletna storitev neodvisna računska enota, ki implementira enotno zahtevo [18]. Primer: odjemalec zahteva atribute nekega natančno določenega objekta, spletna storitev vrne te atribute odjemalcu v obliki XML sporočila.

Spletna storitev:

- je aplikacijska komponenta,
- komunicira s pomočjo odprtih protokolov,
- je samovsebovani *(ang. self-contained)* in samoopisen *(ang. self-describing)*,
- se lahko odkrije oziroma objavi s pomočjo protokola UDDI,
- lahko uporabijo druge aplikacije,
- ima za osnovo protokol HTTP in tehnologijo XML.

Interoperatibilnost ima najvišjo prioriteto. Ko so vse večje platforme lahko dostopale do spleta s pomočjo spletnih brskalnikov, različne platforme niso še bile sposobne komunicirati oziroma sodelovati. Za premostitev te težave so bile razvite spletne aplikacije. Spletne aplikacije so preprosto aplikacije, ki tečejo na spletu. Sledijo standardom spletnih brskalnikov in jih lahko uporabljajo vsi brskalniki na katerikoli platformi.

Spletne storitve dodajo spletnim aplikacijam nov nivo. Spletna aplikacija lahko s pomočjo spletnih storitev objavi svoje funkcije oziroma sporočila vsemu svetu. Spletne storitve za prenos podatkov uporabljajo standard SOAP in za kodiranje oziroma dekodiranje standard XML s pomočjo odprtih

protokolov. S pomočjo spletnih storitev lahko strežniki z različnih platform preprosto komunicirajo med sabo.

Spletne storitve so lahko uporabljene na dva načina:

- kot komponente aplikacije z možnostjo ponovne uporabe kot so na primer pretvorbe merskih enot, vremenska napoved, jezikovni prevodi itn.,
- ali za povezovanje obstoječe programske opreme.

Spletne storitve lahko pripomorejo k interopeatibilnosti tako, da ponujajo različnim aplikacijam sposobnost povezljivosti podatkov kar pomeni, da se z uporabo spletnih storitev lahko izmenjujejo podatki med različnimi aplikacijami in platformami [6].

2.3 MySQL in SQL

MySQL je odprtokodni sistem za upravljanje s podatkovnimi zbirkam [10]. Je med najbolj razširjenimi podobnimi sistemi, po nekaterih ocenah celo najbolj razširjen sistem. Eden izmed razlogov za njegovo razširjenost je zagotovo brezplačnost. Omeniti velja, da je zelo zmogljiv in hiter kar pomeni, da porabi zelo malo sistemskih virov in ga lahko uporabimo celo na starejši in manj zmogljivi strojni opremi. Je tudi povečljiv *(ang. highly scalable)* kar pomeni, da raste skupaj s sistemom katerega podatke upravlja.

Podatkovna zbirka je strukturirana zbirka podatkov. Ta zbirka je lahko enostaven nakupovalni seznam, zbirka slik ali pa obsežna zbirka podatkov korporacij. Za dodajanje, dostopanje do in procesiranje podatkov shranjenih v podatkovni zbirki je potreben sistem za upravljanje podatkovnih zbirk kot je strežnik MySQL. Ker se računalniki dobro izkažejo pri rokovanju z velikimi količinami podatkov, igrajo sistemi za upravljanje podatkovnih zbirk osrednjo vlogo pri računanju kot samostojne enote ali pa kot deli aplikacij.

Relacijske podatkovne zbirke namesto v eni sami veliki tabeli hranijo podatke v ločenih majhnih tabelah. Podatkovne strukture zbirke so organizirane v fizičnih datotekah, ki so optimizirane za hitrejše delovanje. Logični model z objekti kot so podatkovne zbirke, tabele, pogledi, vrstice in stolpci

ponuja gibljivo *(ang. flexible)* programsko okolje. Programer sam zastavi razmerja med različnimi podatkovnimi strukturami kot so številnost (kardinalnost), obveznost in enoličnost, definira pa tudi povezave med tabelami. Podatkovna zbirka sledi tem pravilom kar pomeni, da pri dobro zasnovani podatkovni zbirki aplikacija nikoli ne naleti na nekonsistentne, podvojene, osirotele, neposodobljene ali manjkajoče podatke.

V imenu MySQL kratica SQL pomeni *Structured Query Language* oziroma *strukturirani poizvedovalni jezik*. SQL je najpogostejši standardizirani jezik, ki se uporablja za dostopanje do podatkovnih zbirk. Odvisno od programskega okolja lahko programer uporabi SQL zahtevek neposredno (npr. za izdelavo poročila), vključi SQL zahtevek v programsko kodo drugega programskega jezika ali pa uporabi programskemu jeziku lasten API, ki zakrije SQL zahtevek.

SQL je definiran s standardom ANSI/ISO [13]. SQL standard se razvija od leta 1986, zato obstaja več verzij standarda. SQL-92 se nanaša na standard definiran leta 1992, SQL:1999 se nanaša na standard definiran leta 1999 in SQL:2003 pa se nanaša na najnovejšo različico standarda. Fraza *standard SQL* se običajno nanaša na najnovejšo različico SQL standarda.

MySQL je odprtokodni sistem. Odprtokodno programsko opremo lahko uporablja, spreminja ali pregleduje njeno izvorno kodo kdorkoli. MySQL lahko na svoj stroj s spleta prenese in uporablja kdorkoli popolnoma brezplačno. Če želi, lahko tudi prosto pregleduje njegovo izvorno kodo in jo po potrebi spreminja. Programski produkt MySQL je dostopen pod pogoji splošnega dovoljenja GNU *(ang. GNU General Public License (GPL))*. Licenca določa, kaj sme uporabnik storiti s programsko opremo glede na dano situacijo.

Nekaj določil splošnega dovoljenja GNU:

- Dovoljuje prosto razmnoževanje, urejanje kode, izboljševanje programa. Urejevalec se obveže, da spremembe pošlje avtorju programske kode.
- Dovoljuje prosto urejanje programske kode, vendar mora spremenjena programska koda ostati odprtokodna.

- Programska koda, izdana pod določili GPL, se lahko prosto razmnožuje brezplačno oziroma s plačilom medija in ostalega potrošnega materiala (zgoščenke, embalaža, poštne storitve...)

V primeru, ko uporabniku licenca GPL ne ustreza oziroma želi uporabiti MySQL v komercialne namene, lahko tudi kupi plačljivo različico MySQLa.

Strežnik MySQL lahko brez težav teče že na osebnih ali prenosnih računalnikih poleg ostalih aplikacij, ki jih uporablja običajen uporabnik z minimalno nadzora. MySQL lahko teče tudi na namenskih strežnikih, kjer lahko izkoristi celoten pomnilnik, poljubno število procesorjev in vhodno izhodne sposobnosti strežnika. MySQL je zmožen teči tudi na množici med seboj povezanih strežnikov.

Strežnik MySQL deluje v sistemih odjemalec/strežnik ali vgrajenih *(ang. embedded)* sistemih. Programski paket MySQL je sistem odjemalec/stežnik ki sestoji iz večnitnega strežnika SQL, ki podpira različne ozadne procese *(ang. backends)*, različne odjemalce in knjižnice, administratorska orodja in širok nabor aplikacijskih programskih vmesnikov (API).

Strežnik MySQL se lahko uporabi tudi kot vključena večnitna knjižnica, ki jo uporabnik lahko vključi v svojo aplikacijo in tako dobi manjši, enostavnejši za upravljanje in hitrejši samostojen produkt.

MySQL ponuja tudi pester nabor programske opreme, ki so jo prispevali uporabniki.

2.4 Mobilna platforma Android

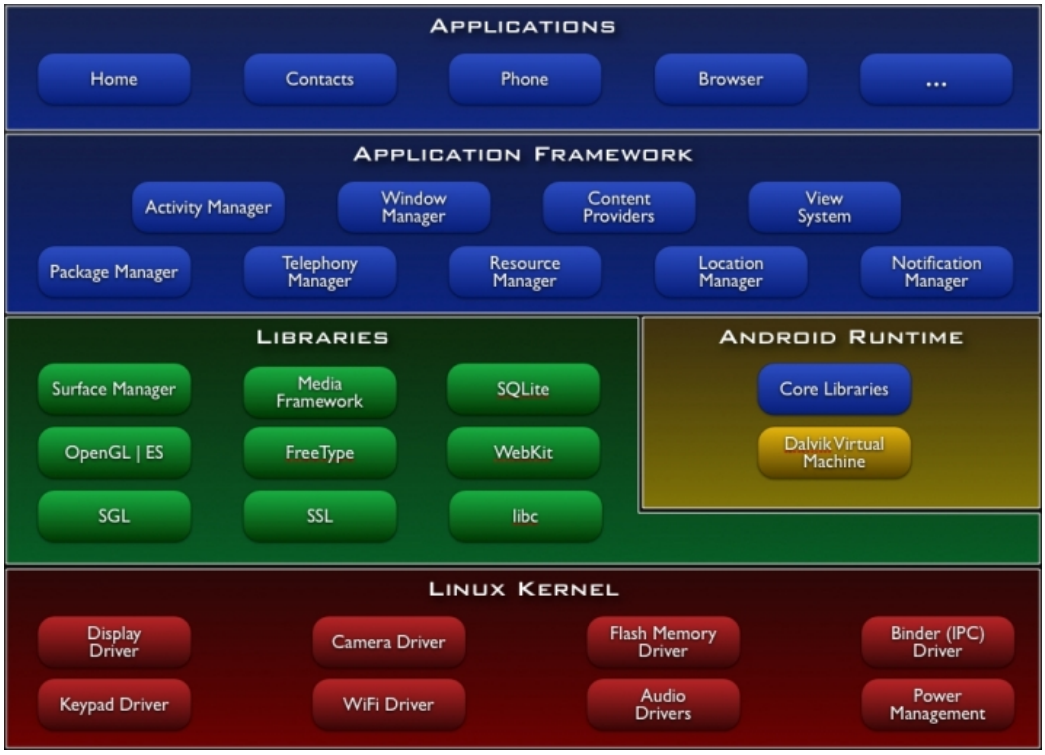
> *Prva resnično odprta in celovita platforma za mobine naprave. Vsebuje vso programsko opremo, potrebna za delovanje mobilnega telefona a brez avtorskih ovir, ki so omejevale inovativnost pri mobilnem razvoju.*
>
> *Google [9]*

Android je odprtokodna mobilna platforma, v prvi vrsti namenjena mo-

bilnim napravam kot so pametni telefoni in tablični računalniki. Vključuje operacijski sistem, vmesno programsko opremo in ključne aplikacije. Najbolj izpostavljana razlika med sistemom Android in konkurenčnimi sistemi je, da je omogočen razvoj kakršnihkoli aplikacij komurkoli vključno z nadomestnimi aplikacijami za privzete aplikacije. Do prodora sistema Android so prenosne naprave poganjale lastniške operacijske sisteme, ki so za razvoj aplikacij zahtevali plačljivo razvojno okolje. Proizvajalci mobilnih naprav so navkljub alternativam svojim napravam prilagali lastne aplikacije, kar se je odražalo na počasnejšem razvoju mobilne tehnologije. Na Androidu so aplikacije razvite z enakimi knjižnicami in se izvajajo na enotnem izvajalnem okolju kar omogoča razvoj kompleksnejših aplikacij, ki zahtevajo zmogljivejšo strojno opremo, kar pa vodi v hitrejši razvoj mobilnih naprav [17], [1].

2.4.1 Arhitektura operacijskega sistema Android

Android sestoji iz več soodvisnih delov na sliki 2.1. Osnova je strojna referenčna zasnova, ki opisuje zahtevane zmogljivosti za mobilne naprave, da lahko podpirajo programski paket. Srčika operacijskega sistema je jedro operacijskega sistema Linux (Linux kernel), ki omogoča interakcijo s strojno opremo na najnižjem nivoju, upravljanje s pomnilnikom in kontrolo procesov, optimizirano za mobilne naprave. Poleg jedra Android vsebuje odprtokodne knjižnice za razvoj aplikacij kot so OpenGL, SQLite, WebKit... in pogonsko okolje za zagon in izvajanje Android aplikacij vključno z Dalvikovim navideznim strojem in jedrnimi knjižnicami, ki aplikacijam omogočajo uporabo funkcionalnosti specifičnih za Android. Pogonsko okolje je prilagojeno in optimizirano za delovanje na mobilnih napravah. Aplikacijsko ogrodje *(ang. framework)* ki omogoča interakcijo med aplikacijskim slojem in sistemskimi storitvami kot so upravljalnik oken *(ang. window manager)*, upravljalnik lokacije *(ang. location manager)*, telefonija, P2P in posredovalec vsebine *(ang. content provider)*, nastopa kot vmesni sloj med knjižnicami in aplikacijami. Ogrodje uporabniškega vmesnika gosti in zaganja aplikacije. Prednaložene aplikacije so dodane kot del osnovnega svežnja in uporabniku



Slika 2.1: Arhitektura operacijskega sistema Android.

ponujajo osnovne funkcionalnosti sistema Android. Kompleta za razvoj programske opreme (SDK) z orodji, vključki in dokumentacijo pa se poslužujejo razvijalci mobilnih aplikacij za sistem Android.

Operacijski sistem Android torej lahko označimo kot sklad različnih slojev, kjer je vsak sloj skupek več programskih komponent. Vsak sloj streže naslednjemu na različne načine. Oglejmo si podrobnejši opis slojev v nadaljevanju.

Jedro Linux

Osnovni sloj predstavlja jedro Linux *(ang. Linux kernel)*. Operacijski sistem Android do vključno verzije 3.x je zgrajen na osnovi jedra Linux 2.6, Android 4.x pa na osnovi jedra Linux 3.4 skupaj z nekaterimi arhitekturnimi spremembami Googla. Linux upravlja s strojno opremo in vsebuje vse nujne

gonilnike strojne opreme. Gonilniki so programi, ki skrbijo za komunikacijo med strojno opremo in operacijskim sistemom. Jedro Linux prav tako deluje kot abstrakcijski sloj med strojno opremo in ostalimi programskimi sloji. Android uporablja Linux za vse svoje jedrne funkcionalnosti kot so upravljanje s pomnilnikom, upravljanje s procesi, varnost, mrežna komunikacija idr. Ker je osnova Androida sistem, ki je dokazano zanesljiv, saj je v uporabi za podporo večini kritičnih aplikacij, njegova popularnost med razvijalci pa na visokem nivoju, prestavitev na različne strojne osnove ne predstavlja večjih težav.

Knjižnice

Naslednji sloj predstavljajo Androidove knjižnice *(ang. native libraries)*. Ta sloj omogoča napravi, da uporablja različne tipe podatkov. Knjižnice so pisane v jezikih C ali C++ in so specifične za določeno strojno opremo.

Knjižnica *Surface Manager* se uporablja pri sestavljanju okenskega upravljalca s pomočjo pomožnega pomnilnika *(ang. off-screen buffering)*. Pomožni pomnilnik se uporablja, ker ni mogoče risati neposredno na prikazan zaslon, elementi za izris se namesto tega zapišejo v pomožni pomnilnik. Tam se novi elementi dodajo že obstoječim in se nato združijo v zaslonsko sliko, ki se prikaže uporabniku. Pomožni pomnilnik tako omogoča prosojne okenske poglede.

Knjižnica *Media framework* ponuja različne kodeke ter omogoča kreiranje in predvajanje različnih formatov zapisa. Knjižnica *SQLite* je stroj za upravljanje podatkovnih zbirk *(ang. database engine)* in je namenjen upravljanju s podatki. Knjižnica *WebKit* je brskalniški stroj *(ang. browser engine)* namenjen prikazovanju HTML vsebin. Knjižnica *OpenGL* omogoča izris dvodimenzionalne in tridimenzionalne grafike na zaslon.

Izvajalno okolje Android

Izvajalno okolje Android *(ang. Android Runtime)* sestoji iz Dalvikovega navideznega stroja *(ang. Dalvik Virtual Machine)* in jedrnih knjižnic Java *(ang.*

Core Java libraries).

Dalvikov navidezni stroj je tip navideznega stroja Java *(ang. Java Virtual Machine)*, ki se uporablja na androidnih napravah za poganjanje aplikacij. Optimiziran je za izvajanje v okoljih, ki zahtevajo nizko porabo energije ter majhno porabo procesne moči in pomnilnika. Za razliko od JVM Dalvikov virtualni stroj ne poganja datotek, ki vsebujejo Javansko bitno kodo (*.class), ampak optimizirane kompaktne datoteke, zgrajene iz Javanske bitne kode (*.dex). Te datoteke so zgrajene med prevajanjem in so bolj učinkovite v okoljih z omejenimi viri. Dalvikov virtualni stroj dopušča izvajanje več primerkov virtualnih strojev hkrati in tako poskrbi za varnost, neodvisnost, upravljanjes pomnilnikom in podporo večnitnosti. Dalvikov virtualni stroj je razvil Dan Bornstein iz podjetja Google.

Jedrne knjižnice Java se razlikujejo od knjižnic Java SE in Java ME, vendar ponujajo večino funkcionalnosti definiranih v knjižnicah Java SE.

Aplikacijsko ogrodje

Aplikacijsko ogrodje sestoji iz blokov, s katerimi aplikacije sodelujejo neposredno. Blok je zaključena celota, ki upravlja z določeno funkcionalnostjo aplikacijskega ogrodja. Na primer *Activity Manager* ki sodeluje z aktivnostmi, ki tečejo na sistemu. Ti programi upravljajo z osnovnimi funkcijami naprave kot sta na primer upravljanje z viri in upravljanje telefonskih klicev. Razvijalec lahko gleda na bloke kot na skupek orodij, s katerimi gradi svojo aplikacijo.

Pomembnejši bloki aplikacijskega ogrodja so:

- *upravljalec aktivnosti (ang. Activity Manager)* upravlja z življenjskim ciklom aplikacij,
- *posredovalci vsebine (ang. Content Providers)* upravljajo z deljenjem podatkov med aplikacijami,
- *upravljalec telefonije (ang. Telephony Manager)* upravlja telefonske klice. Ko želi programer implementirati lastno aplikacijo za sprejemanje telefonskih klicev uporabi tega upravljalca,

- *upravljalec lokacije (ang. Location Manager)* upravlja z lokacijami s pomočjo GPSja,
- *upravljalec virov (ang. Resource Manager)* upravlja z različnimi vrstami virov ki jih lahko uporabijo aplikacije.

Aplikacije

Aplikacije so vrhnji sloj arhitekture Android kamor spada tudi aplikacija, ki je izdelana v okviru tega diplomskega dela. Skupaj z operacijskim sistemom uporabnik dobi tudi nekaj prednaloženih aplikacij kot so odjemalec SMS sporočil, spletni brskalnik, aplikacija za telefonijo in upravljalec stikov.

Razvijalci imajo torej možnost razviti aplikacije, ki nadomestijo prednaložene aplikacije in niso omejeni z nobeno lastnostjo *(ang. feature)* operacijskega sistema. Drugače rečeno so razvijalci neomejeni z razvojem dokler uporabnik aplikaciji dovoli uporabo željenih lastnosti in tako Android razvijalcu ponuja neomejene priložnosti.

2.5 Arhitektura REST

Poenostavljeno rečeno je REST (**Re**presentational **S**tate **T**ransfer) skupek omejitev, ki posplošijo arhitekturne elemente znotraj distribuiranega hipermedijskega sistema. Posledica upoštevanja teh omejitev je arhitektura, ki je razširljiva, uporabna in dostopna. Kot vse trditve na tej ravni posplošitve ta način določanja arhitekture ni dokazljiv, vendar po izkušnjah sledenje temu slogu vodi do manj odvisnih modulov načrtovanega sistema drug od drugega [18], [15].

Arhitekturni slog REST se lahko uporablja tudi pri razvoju spletnih storitev kot alternativa ostalim razpršenim računskim specifikacijam *(ang. distributed-computing specifications)* kot je SOAP.

Lastnosti arhitekturnega sloga REST so:

- zmogljivost,
- možnost razširitve interakcije komponent,

- enostavnost vmesnikov,
- prilagodljivost komponent da sledijo spremenljivim potrebam, tudi med izvajanjem aplikacije,
- vidnost komunikacije med komponentami storitvenih sredstev *(ang. service agents)*,
- prenosljivost namestitve komponent *(ang. component deployment)*,
- zanesljivost.

2.5.1 Omejitve

Arhitekturne lastnosti REST so posledica natanko določenih omejitev pri komponentah, povezavah in podatkovnih elementih.

Formalne omejitve REST so:

Odjemalec-strežnik

Enoten vmesnik loči ojemalce od strežnikov. Ločitev pomeni, da se odjemalec ne zaveda podatkovne zbirke, ko zahteva podatke. S podatkovno zbirko upravlja samo strežnik interno. Posledično je izboljšana prenoslivost odjemalčeve kode, strežniki pa ne vsebujejo uporabniškega vmesnika in so zato enostavnejši in bolj razširljivi. Odjemalci in strežniki se lahko razvijajo ali nadomestijo neodvisno drug od drugega, dokler je med njimi zagotovljen isti vmesnik.

Brezstanjskost

Komunikacija med odjemalcem in strežnikom je omejena z odsotnostjo stanja konteksta *(ang. stateless context)* odjemalca na strežniku med zahtevki. Vsak zahtevek odjemalca vedno vsebuje vse informacije, ki so potrebne za streženje zahtevku. Stanje seje beleži odjemalec. Pomembno je vedeti, da lahko stanje seje strežnik dostavi nekemu drugi storitvi kot je na primer podatkovna zbirka, da zagotovi trajno stanje za nekaj časa in tako omogoči avtentikacijo. Odjemalec lahko tako prične z izdajanjem zahtevkov, ko je pripravljen na vstop v novo stanje seje. Ko odjemalec izda enega ali več zahtevkov se šteje, da je v tako imenovanem prehodnem stanju *(ang. in transition)*. Reprezenta-

cija vsakega stanja aplikacije vsebuje povezave, ki so lahko aktivirane ob naslednjem prehodu odjemalca v novo stanje.

Predpomnjenje

Odjemalci na spletu lahko odziv na zahtevek predpomnijo. Posledično mora za odzive eksplicitno ali implicitno veljati, ali so lahko predpomnjeni ali ne, da se odjemalci izognejo uporabi neposodobljenih podatkov pri izdajanju novih zahtev. Dobro izvedeno predpomnenje delno ali popolnoma odstrani potrebo po nekaterih stikih odjemalca s strežnikom, kar še izboljša povečljivost in zmogljivost.

Večplastnost

Odjemalec ne more zanesljivo ugotoviti, ali je neposredno povezan na ciljni strežnik, ali pa komunicira s posrednikom. Posredovalni strežniki lahko izboljšajo razširljivost z omogočanjem izravnave *(ang. load balancing)* obremenitve in ponujanjem deljenega predpomnilnika *(ang. shared cache)*.

Izvršna koda na zahtevo (izbirno)

Strežniki lahko začasno razširijo ali prilagodijo funkcionalnost odjemalca s prenosom izvršne kode. Primeri te prakse so lahko prevedene komponente kot so apleti Java in skripte odjemalca kor je na primer JavaScript. To je tudi edina izbirna omejitev arhitekture REST.

Enoten vmesnik

Enoten vmesnik je temeljna omejitev pri načrtovanju storitev REST. Enoten vmesnik poenostavi in jasno razmeji arhitekturo kar omogoči samostojen razvoj vsakega dela sistema. Štirje vodilni principi takšnega vmesnika so:

Opredelitev virov: Posamezni viri so opredeljeni v zahtevah, na primer URI pri medmrežnih storitvah REST. Sami viri so konceptualno ločeni od predstavitev, ki jih prejme odjemalec. Za ilustracijo, strežnik ne odgovori z rezultatom poizvedbe podatkovne zbirke ampak v odjemalcu prijazni obliki, na primer z JSON objektom, HTML ali XML dokumentom ki vsebujejo rezultate poizvedbe iz podatkovne zbirke.

Upravljanje z viri glede na obliko odgovora: Ko odjemalec poseduje predstavitev vira vključno z morebitnimi metapodatki ima dovolj informacije, da lahko spremeni ali izbriše vir na strežniku, če ima dovoljenje.

Samoopisna sporočila: Vsako sporočilo vključuje dovolj informacije o načinu obravnave tega sporočila. Na primer postavka *Media type* določi, kateri razčlenjevalnik *(ang. parser)* naj se uporabi pri razčlenjevanju sporočila. Odgovori tudi eksplicitno določijo, ali dopuščajo predpomnjenje.

Hipermedij (ang. hypermedia) kot upravljalec stanja aplikacije Odjemalci prehajajo med stanji samo preko dejanj, ki jih strežnik dinamično prepozna znotraj hipermedijev (npr. hiperlinki v hipertekstu). Razen enostavnih vstopnih točk aplikacije odjemalec ne privzame, da je na voljo kakršna koli akcija za določen vir razen tistih, ki so opisani v odgovoru, ki je bil prej prejet s strani strežnika.

Aplikacije, ki sledijo opisanim omejitvam, se lahko imenujejo polno REST aplikacije *(ang. RESTful)*. Če aplikaciji manjka le ena izmed naštetih lastnosti, ta aplikacija ni polno REST aplikacija. Vključevanje teh omejitev in s tem prilagoditev arhitekturnemu slogu REST omogoča kakršnemu koli porazdeljenemu hipermedijskemu sistemu, da ima zaželjene lastnosti kot so zmogljivost, povečljivost, preprostost, prilagodljivost, prepoznavnost, prenosljivost in zanesljivost.

2.5.2 Kako uporabiti arhitekturo REST pri snovanju spletnih storitev

Spletne storitve, ki sledijo omejitvam REST, se imenujejo spletne storitve REST *(ang. RESTful Web services)*. Tipičen zahtevek REST izgleda takole:

- osnoven URI npr. *http://brest.si/EkoBuMDesign/viri/*,
- vrsta podatkov *(ang. media type)*. To je pogosto JSON objekt, vendar ga lahko nadomesti katerikoli druga vrsta podatka kot so XML, Atom,

slike, video idr.,

- standardne HTTP metode GET, PUT, POST ali DELETE katerih delovanje je prikazano v tabeli 2.1,
- povezave v hipertekstu do referenčnih stanj,
- povezave v hipertekstu do virov, ki jih zahtevajo ustrezna stanja.

	Skupina virov	Posamezni vir
Primer URIja	http://brest.si/naslStrank/	http://brest.si/naslStrank/stranka17
GET	Pridobi seznam virov	Pridobi natanko ta vir
PUT	Zamenjaj seznam virov	Zamenjaj natanko ta vir
POST	Naloži seznamvirov (na cilj)	Naloži natanko ta nov vir (na cilj)
DELETE	Izbriši seznam virov	Izbriši natanko ta vir

Tabela 2.1: Tabela prikazuje običajno uporabo metod HTTP za implementacijo vmesnika REST in opise, kaj se zgodi pri posamezni metodi glede na izbrano količino virov, do katerih REST zahtevek želi dostopati.

Za vmesnike REST uradni standard ne obstaja, saj je REST le arhitekturni slog. Kot alternativa se lahko uporablja protokol SOAP, za katerega pa standard obstaja. Čeprav REST ni protokol, lahko še vedno uporablja ostale standarde kot so HTTP, XML, URI itd [11].

Poglavje 3

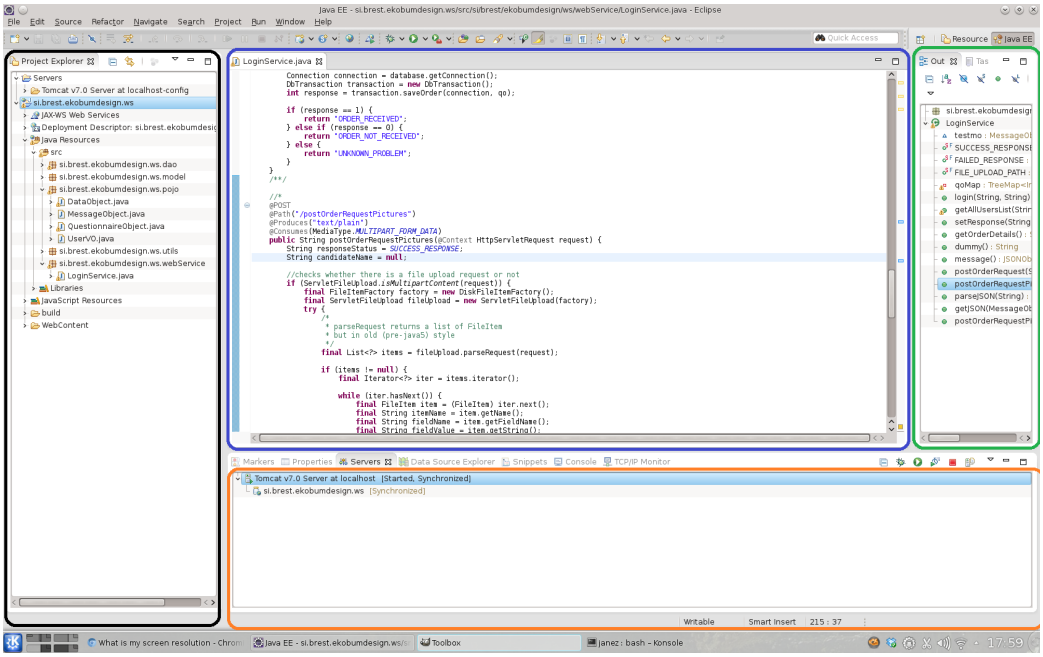
Razvojna orodja

Za izdelavo prototipa rešitve problema, s katerim se ukvarja ta naloga, so bila uporabljena nekatera bolj razširjena oziroma splošna orodja. Za razvoj mobilne aplikacije je bilo uporabljeno orodje Android SDK, ki že vključuje Eclipse in za razvoj spletne storitve brezplačno razvojno okolje Eclipse.

3.1 Eclipse

Eclipse je integrirano razvojno okolje *(ang. integrated development environment - IDE)*, ki obsega delovni prostor *(ang. workspace)* in vtičnike *(ang. plugins)*, ki omogočajo prilagajanje in razširitve okolja. Večinoma je pisan v programskem jeziku Java. Uporablja se za razvoj aplikacij v Javi in s pomočjo vključkov tudi v drugih programskih jezikih kot so na primer Ada, C, C++, COBOL, Fortran, Perl, PHP in še mnogih drugih [14].

Eclipse IDE, ki je izdan pod pogoji Eclipse Public License, je brezplačna odprtokodna programska oprema. Osnovni pogled na sliki 3.1 prikaže najpomembnejše poglede *(ang. views)* kot so urejevalnik kode (moder okvir), pregledovalnik paketov (črn okvir), povzetek kode(zelen okvir) in beležka. Pri razvoju spletne storitve je bil uporabljen Eclipse Juno Service Release 2.



Slika 3.1: Tipični pogled razvojnega okolja Eclipse, s pomočjo katerega je bila razvita spletna storitev.

3.2 Razvojno okolje Android

Paket za razvoj programske opreme Android *(ang. Android Software Development Kit)* vsebuje bogat nabor razvojnih orodij za razvoj mobilnih aplikacij na mobilni platformi Android. Med drugim vsebuje razhroščevalnik, knjižnice, emulator mobilnih naprav, dokumentacijo, primere delujoče kode in predloge. Uradno podprto razvojno okolje za Android SDK je Eclipse, vendar je razvoj mogoč tudi v drugih okoljih, na voljo pa je tudi razvoj samo s pomočjo ukazne vrstice. Android SDK prav tako omogoča razvoj aplikacij za starejše verzije operacijskega sistema in nudi možnost prenosa dodatnih komponent na zahtevo razvijalca.

Slika 3.2 prikazuje osnovno okno razvojenga okolja, kjer so vidni najpomembnejši pogledi urejevalnik kode (moder okvir), pregledovalnik paketov (črn okvir), povzetek kode (zelen okvir), razhroščevalnik (oranžen okvir) in emulator mobilnih naprav (rdeč okvir).

Aplikacija Android je pakirana v formatu APK *(ang. Android application package file)* in shranjena v mapi */data/app*. Ta mapa je dostopna samo s korenskim dostopom zaradi varnostnih razlogov. Paket APK vsebuje datoteke DEX, ki so prevedena bitna koda oziroma Dalvik zagonske datoteke, multimedijske datoteke itd.

Pomembnejše funkcionalnosti so:

integrirano ustvarjanje, razhroščevanje, pakiranje, nameščanje in grajenje projekta Android,

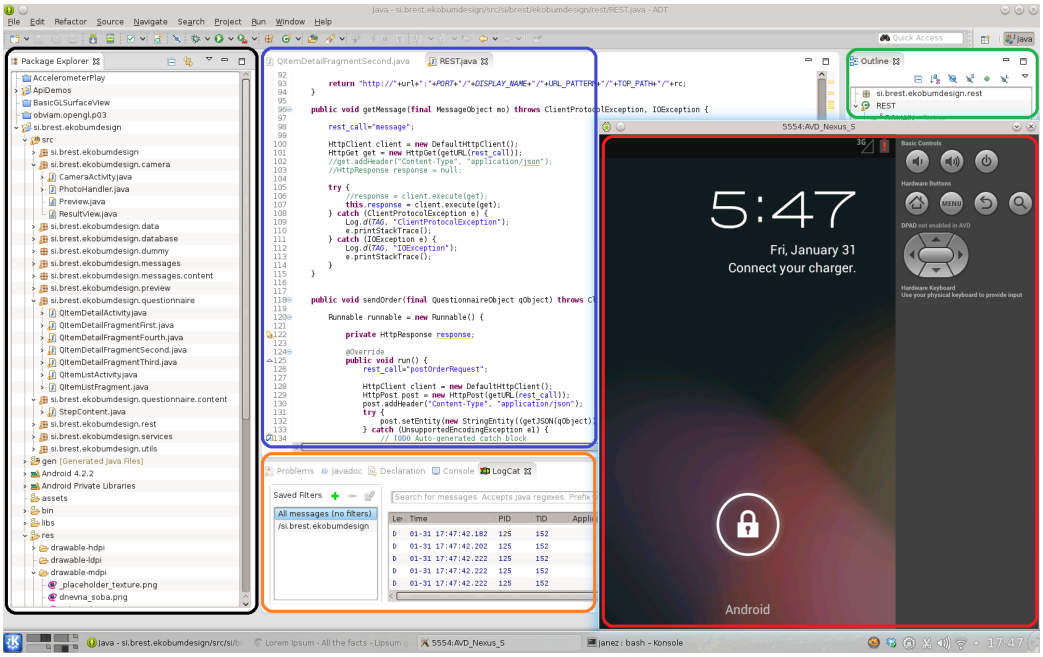
Več nalog razvojnega delovnega procesa ADT vključi v okolje Eclipse in tako zagotovi lažji in hitrejši razvoj ter testno okolje.

integracija orodij SDK,

Veliko orodij SDK paketa je vgrajenih v izbirno vrstico okolja Eclipse, njegove poglede ali pa tečejo v ozadju.

programski jezik Java in urejevalniki XML,

Urejevalnik programskega jezika java vsebuje najpogostejše značilnosti integriranih razvojnih okolij kot je pregledovanje sintakse med prevajanjem, samodejno dokončanje med pisanjem in vgrajena dokumentacija



Slika 3.2: Tipični pogled orodja za razvoj aplikacij Android.

okvirja aplikacijskega vmesnika Android. Razvojno okolje prav tako ponuja lastne urejevalnike XML, ki omogočajo urejanje Androidu lastnih formiranih datotek XML. Grafični vmesnik omogoča oblikovnaje uporabniških vmesnikov s pomočjo datotek XML.

in vgrajena dokumentacija okvirja aplikacijskega vmesnika Android.

Dokumentacija je dostopna z držanjem kazalca nad imeni razredov, metod in spremenljivk.

Poleg izbranih orodji na trgu obstaja veliko število razvojnih okolij, ki so ravno tako dovolj zmogljiva za realizacijo zastavljenega roblema. Ključni dejavniki pri izbiri uorabljenih orodij so odprtokodnost, široka baza uporabnikov, ki hitro ponudijo odgovor pri morebitnih zapletih, dejaven razvoj okolja, možnost uporabljanja razširitev in veliko obstoječih primerov rešitev podproblemov na sletu.

Poglavje 4

Razvoj sistema EkoBuMDesign

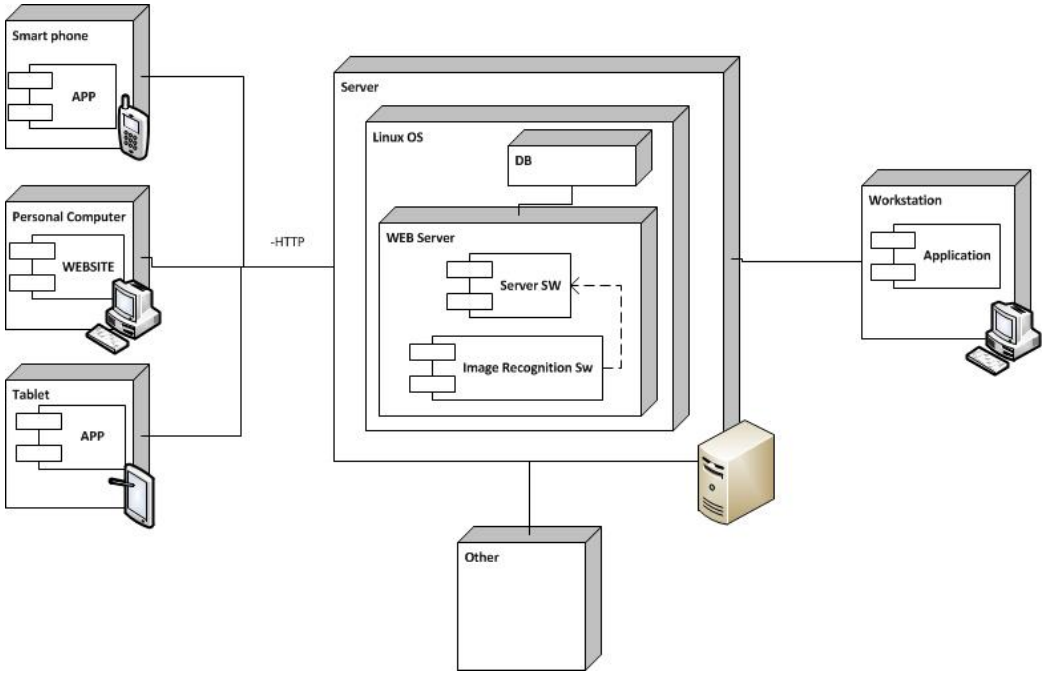
V tem poglavju je predstavljen rezultat diplomskega dela. V prvem delu je predstavljen celoten načrtovan sistem. Potem sledi predstavitev posameznih delov sistema, ki so bili realizirani.

Prva je mobilna aplikacija EkoBuMDesign, ki deluje kot odjemalec. EkoBuMDesign je akronim projekta. S to aplikacijo rokuje stranka, s pomočjo katere naroči storitev podjetja in si interaktivno ogleduje rezultat dela arhitekta. Za razumevanje delovanja aplikacije je potrebno razumevanje osnov delovanja aplikacij na operacijskem sistemu Android, zato se poglavje najprej osredotoči na le te.

Nato je predstavljena spletni storitev, ki skrbi za povezljivost med strežnikom in odjemalci - mobilnimi aplikacijami. Na koncu je predstavljena podatkovna zbirka MySQL, ki skrbi za aktualnost podatkov na strani strežnika.

4.1 Arhitektura sistema

Slika 4.1 predstavlja arhitekturo celotnega sistema. Osnova sistema je spletni strežnik, na katerem teče spletna storitev sistema. Spletna storitev komunicira s podatkovno zbirko, kamor shranjuje vse spremembe, ki so lahko ustvarjanje uporabniškega računa, prejem naročila, prejem oziroma pošiljanje sporočila idr. Spletni strežnik teče na strojni opremi podjetja.

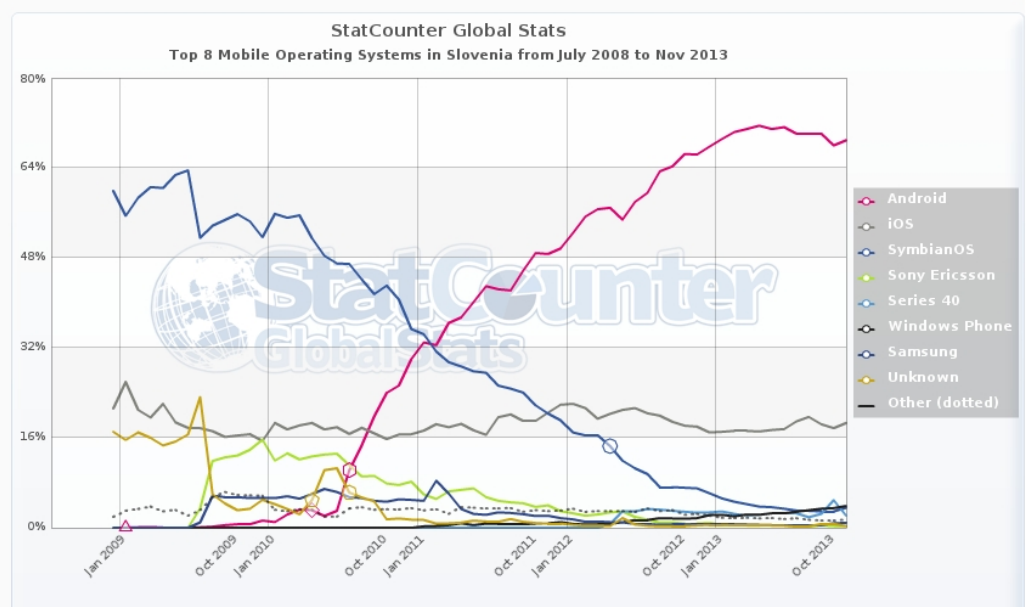


Slika 4.1: Arhitektura sistema.

Na strežniku podjetja teče tudi programska rešitev za izdelavo tridimenzionalnega prostora s pomočjo fotografij, vendar se s tem problemom diplomska naloga ne ukvarja. Mobilna aplikacija tako le privzame, da je bila ta operacija na strežniku opravljena in pričakuje izrisane teksture oblikovanega prostora, ki jih nato na zahtevo stranke zmore prikazati.

Stranke imajo na voljo mobilno aplikacijo, ki jo prenesejo na svojo mobilno napravo. Aplikacija komunicira s strežnikom podjetja s pomočjo protokola HTTP, vse zahteve pa sledijo arhitekturnemu slogu REST.

Arhitekt na strani podjetja uporablja programsko opremo, s pomočjo katere sprejema naročila. Za naročilo, ki vsebuje vse obvezne podatke, izdela vizualizacijo prostora. Rezultat izvozi v formatu, ki ga mobilna aplikacija lahko prikaže. Rezultat predstavlja tridimenzionalno vizualizacijo prostora stranke, ki ga arhitekt pošlje kot del sporočila stranki. Sporočilo lahko vsebuje tudi tekst, slike in ostale multimedijske predstavitve. Poleg predvidene programske opreme arhitekt uporablja oblikovalska orodja, ki niso predmet



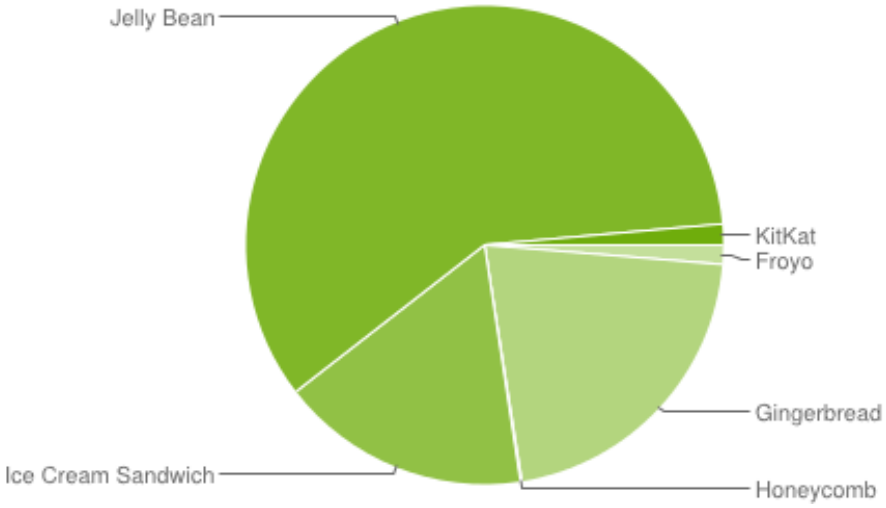
Slika 4.2: Podatki o razširjenosti mobilnih operacijskih sistemov v Sloveniji od junija 2008 do novebra 2013 [8].

zastavljenega problema, ampak jih priskrbi podjetje. Posebno pozornost bo potrebno posvetiti združljivosti podatkov med obstoječimi orodji in novo razvito programsko opremo.

Zaradi zagotavljanja čim nižjih stroškov so bile pri realizaciji uporabljene odprtokodne rešitve. Za spletni strežnik sistema je bil izbran Apache Tomcat, za upravljanje podatkovne zbirke pa sistem MySQL. Oba sta nameščena na operacijskem sistemu z jedrom Linux. Mobilna aplikacija teče na operacijskem sistemu Android.

4.2 Mobilna aplikacija kot odjemalec

Operacijski sistem Android je na Slovenskem z naskokom najbolj razširjen mobilni operacijski sistem (Slika 4.2). To je bil ključni dejavnik pri izbiri mobilne platforme, zaradi česar mobilna aplikacija teče na napravah z operacijskim sistemom Android.



Slika 4.3: Delež verzij operacijskega sistema Android na mobilnih napravah Android [2].

Na trgu obstaja več izdaj operacijskega sistema Android. V času izdelave projekta je najbolj razširjena verzija 4.x, ki vključuje verzije *Ice Cream Sandwich* (4.0 - 4.0.4), *Jelly Bean* (4.1 - 4.3) in *KitKat* (4.4) na sliki 4.3 [7]. Ker verzija 4.x teče na več kot treh četrtinah napravah z operacijskim sistemom Android, je ta verzija tudi minimalna zahtevana za uporabo mobilne aplikacije. Po preizkušanju prototipa mobilne aplikacije so se tablični računalniki izkazali za primernejše medije za uporabo aplikacije, saj je le ta bogata z multimedijskimi vsebinami, ki pridejo bolj do izraza na večjih površinah, vendar aplikacija deluje tudi na moblnih telefonih.

4.2.1 Arhitektura aktivnosti

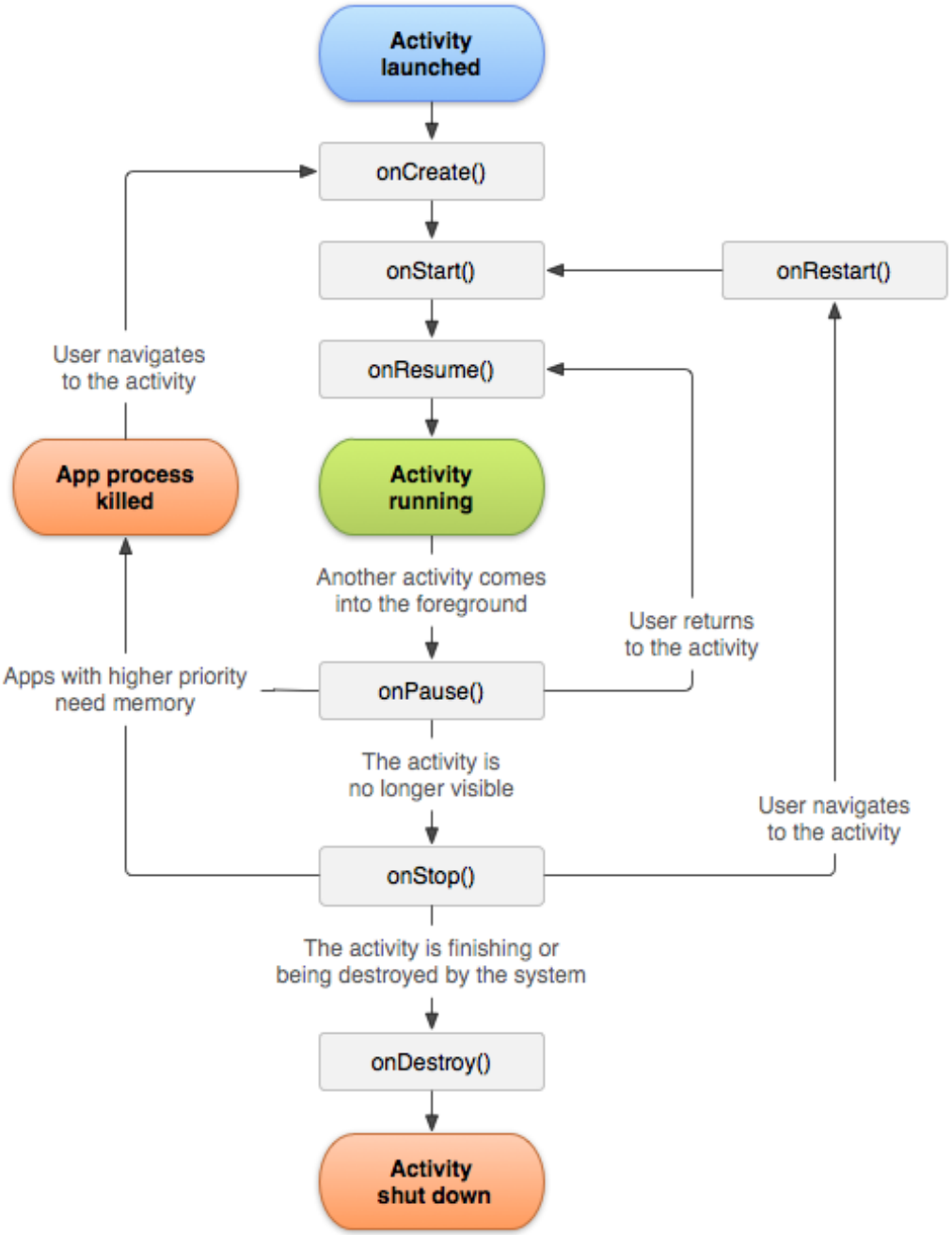
Aktivnost *(ang. Activity)* je tisto, kar uporabnik trenutno vidi na zaslonu. Ta trditev je nekoliko nenatančna, vendar za osnovno razumevanje zadostuje. Aktivnost je del aktivnostnega sklada *(ang. activity stack)*, kjer so na skladu aktivnosti v različnih stanjih. Ker je aktivnost osnovni gradnik mobilne aplikacije je na mestu, da je tu podrobneje predstavljena. Slika 4.4 prikazuje

diagram življenjskega cikla aktivnosti s štirimi stanji in procedurami, ki se izvedejo med prehajanji med njimi:

- Aktivnost se zažene *(ang. Activity launched.* To se lahko zgodi na primer ob ukazu uporabnika z izbiro ustreznega gumba. Izvedejo se procedure *onCreate(), onStart()* in *onResume()*. Nato je aktivnost v ospredju na zaslonu in je na vrhu sklada aktivnosti. Ta aktivnost je aktivnost v teku *(ang. Activity running)*.
- Če aktivnost ni več v žarišču *(ang. in focus)* vendar je še vedno vidna (primer, ko se nad aktivnostjo pojavi prosojna aktivnost, ki je tedaj v žarišču), je takšna aktivnost v premoru *(ang. paused)*. Takrat se izvede procedura *onPause()*. Aktivnost v premoru je še vedno živa kar pomeni, da obdrži vsa stanja in sistemske vire, vendar jo operacijski sistem v primeru kritičnega pomanjkanja virov lahko zaključi *(ang. App process killed)*.
- Če je aktivnost popolnoma zakrita z neko drugo aktivnostjo, je aktivnost v mirovanu *(ang. stopped)*. Še vedno obdrži vsa stanja in sistemske vire vendar ker uprabniku ni vidna je zelo verjetno, da jo operacijski sistem zaključi v primeru pomanjkanja sistemskih virov.
- Če je aktivnost ustavljena ali v premoru, ji lahko sistem pošlje signal za zaključitev, ali pa jo zaključi sam in tako sprosti sistemske vire. Ko jo uporabnik ponovno zažene, mora aktivnost znova obnoviti vsa stanja in pridobiti vse vire.
- Če se aktivnost zaustavi oziroma jo zaustavi operacijski sistem je ta aktivnost zaustavljena *(ang. Activity shut down)*.

V življenjskem ciklu aktivnosti so bistvenega pomena trije cikli:

- Celotna življenjska doba aktivnosti se zgodi med prvim klicem metode *onCreate(Bundle)* in končnim klicem funkcije *onDestroy()*. Aktivnost izdela vsa globalna stanja v funkciji *onCreate()* in sprosti vse vire v funkciji *onDestroy()*. Na primer če aktivnost zažene nit, ki v ozadju prenaša podatke s strežnika, to nit ustvari v funkciji *onCreate()* in ustavi v funkciji *onDestroy()*.



Slika 4.4: Diagram aktivnosti aplikacije Android in njenih stanja [12].

- Vidna življenjska doba aktivnosti se prične med klicem metode *onStart()* in njej ustrezne metode *onStop()*. Med tem časom lahko uporabnik aktivnost vidi na zaslonu, ni pa nujno, da uporabnik rokuje s to aktivnostjo. Med izvedbo teh dveh metodama se zasedejo viri, ki so potrebni za prikaz aktivnosti uporabniku. Na primer v funkciji *onStart()* se prijavi *(ang. register) BroadcastReceiver* ki spremlja spremembe, ki se odražajo na uporabniškem vmesniku in odjavi *(ang. unregister)* v funkciji *onStop()*, ko uporabnik aktivnosti ne vidi več. Torej ob klicu metode *onStart()* se potrebni viri za prikaz zasedejo, ob klicu metode *onStop()* pa se zasedeni viri za prikaz aktivnosti sprostijo. Ti dve metodi se lahko kličeta tudi večkrat, hkrati ko uporabnik aktivnost prikaže na zaslon ali pošlje v ozadje. Viri za prikaz aktivnosti niso nujno vsi potrebni viri aktivnosti, so le del vseh virov, ki jih aktivnost lahko zasede.
- Izvajanje aktivnosti v ospredju se zgodi med klicema procedur *onResume()* in ustreznice *onPause()*. V tem času je aktivnost nad ostalimi aktivnostmi v aktivnostnem skladu in jo uporabnik vidi na zaslonu. Aktivnost lahko pogosto prehaja med aktivnim stanjem in stanjem v premoru. Na primer procedura *onPause()* za aktivnost v teku se izvede, ko naprava preide v spalni način *(ang. sleep mode)* in se takrat zaslon običajno zatemni, ko se izda namera *(ang. intent)* za zagon nove aktivnosti. Obratno pa se procedura *onStart()* izvede, ko uporabnik proceduro v mirovanju prikliče nazaj na zaslon, kjer postane aktivna oziroma je zopet v teku. Prehod med stanjem v teku in stanjem mirovanja mora biti kar se da hiter, zato naj koda v teh funkcijah ne bo zahtevna.

Zaradi razvoja za tablične računalnike in telefone so poleg aktivnosti v aplikaciji uporabljeni tudi fragmenti *(ang. Fragment)*. Fragmenti omogočajo boljšo prilagodljivost uporabniškega vmesnika, ko je ta narejen za naprave z različno velikostjo delovne površine. Tako imajo mobilni telefoni površino mnogo manjšo od tabličnih računalnikov, vendar se prikaz aplikacije s pomočjo

fragmentov prilagaja glede na prikazno površino naprave. Fragmenti imajo podoben življenjski cikel aktivnostim, osnovna razlika pa je v zasedanju prikazne površine. Aktivnost tako vedno zasede celotno površino zaslona, fragment pa samo del. Fragment je tudi vedno del krovne aktivnosti, ki upravlja z življenjskim ciklom vseh lastnih fragmentov.

Na primer ko želimo izdelati aplikacijo, ki ima več zavihkov, jo lahko izdelamo za različne naprave različno glede na površino zaslona. Tako bomo na mobilnih telefonih zavihke prikazali kot seznam v eni aktivnosti, izbira zavihka pa bo aktivnost seznama nadomestila z novo izbrano aktivnostjo. Na tabličnem računalniku pa bomo zavihke prikazali kot seznam v fragmentov na levi strani zaslona, vsaka izbira zavihka pa bo prikazala fragment na desni strani zaslona, pri čemer bo fragment s seznamom na levi strani zaslona še vedno viden.

Na kratko, aktivnost zaseda celotno površino, fragment pa samo del, vendar so vedno del krovne aktivnosti. Ravno tako se življenjski cikel krovne aktivnosti neposredno odraža na življenjskih ciklih vseh hčerinskih fragmentov. Prvič se fragmenti pojavijo pri operacijskem sistemu Androidi 3.0.

4.2.2 Podatkovna zbirka SQLite

Podatkovna zbirka SQLite *(ang. SQLite database)* je odprtokodna podatkovna zbirka. Podpira standardne lastnosti relacijskih podatkovnih zbirk kot so jezik SQL, transakcije in pripravljene izjave *(ang. prepared statements)*. SQLite je vdelan v vsaki napravi z operacijskim sistemom Android. Ne potrebuje nastavitvenih procedur ali administracije podatkovne zbirke, potrebno je le izdelati pravilne zahtevke SQL za tvorjenjenje in osveževanje, nato s podatkovno zbirko operacijski sistem upravlja samodejno. Dostop do podatkovne zbirke vključuje dostop do datotečnega sistema, zato je lahko ta operacija počasna. Priporočljivo je torej to operacijo izvesti asinhrono. Podatkovna zbirka je shranjena v mapi *DATA/data/APP_NAME/databases/FILENAME*. Pot do te lokacije je skonstruirana takole: *DATA* je pot, ki jo vrne klic metode *Environment.getDataDirectory()*, *APP_NAME* je ime mobilne aplika-

cije, *FILENAME* pa ime podatkovne zbirke, pri razviti mobilni aplikaciji je to *Brest*.

4.2.3 Aktivnosti mobilne aplikacije

Načrtovanje mobilne aplikacije sledi priporočilom Googlovih razvijalcev. Celotna shema mobilne aplikacije je opisana po korakih, po katerih je nastajala.

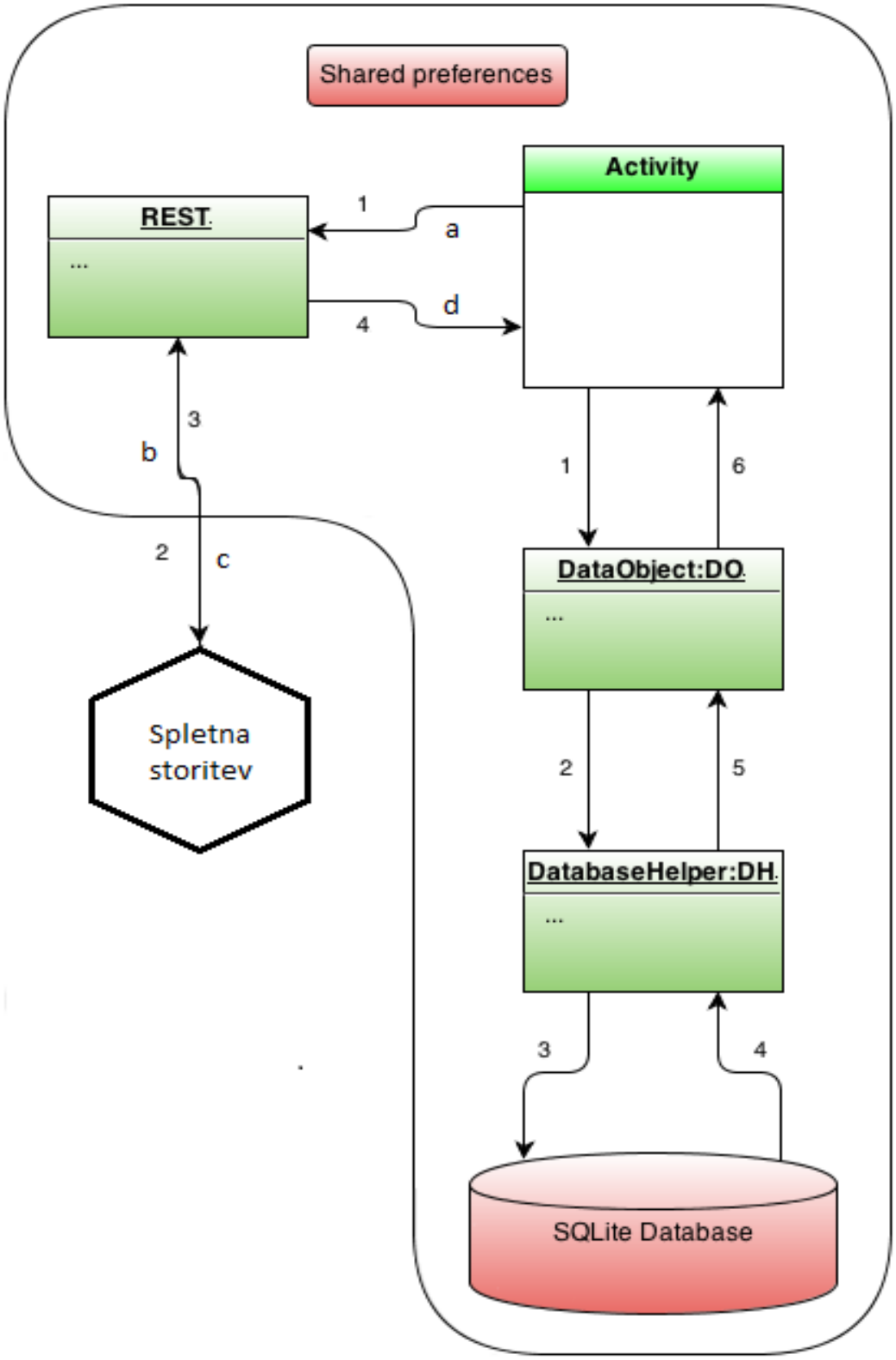
Tekom razvoja se je pokazala potreba, da se mobilno aplikacijo obravnava v dveh delih, z aktivnostjo *(ang. activity)* na sliki 4.5 in storitvijo v ozadju *(ang. background service)* na sliki 4.6. V prvem delu stranka z aplikacijo upravlja neposredno, v drugem pa storitev mobilne aplikacije samodejno posreduje informacije strežniku in jih od njega pridobiva. Storitev mobilne aplikacije se zažene ob zagonu naprave, na kateri teče operacijski sistem Android in za osveževanje podatkov ni potrebno, da stranka storitev zažene ročno z zagonom aplikacije. Z drugimi besedami operacijski sistem storitev aplikacije zažene samodejno ob zagonu oziroma vzpostavitvi povezave z internetom oziroma zaustavi ob prekinitvi povezave s spletom.

Elementi na sliki 4.5, ki so obkroženi z okvirjem, sodijo v kontekst aplikacije, z njimi aplikacija neposredno upravlja.

Razvita mobilna aplikacija je sestavljena iz več aktivnosti. Sledijo si po hierarhiji z osnovnim menijem na vrhu, preko katerega stranka požene eno izmed ostalih aktivnosti. Načrtovane aktivnosti so:

- osnovni meni *(Main Activity)*, ki stranki služi kot vstopna točka,
- podaktivnosti vprašalnik *(Questionnaire Activity)*, kjer stranka izpolni vprašalnik o prostoru ob naročilu storitve,
- nastavitveno stran *(Preferences Activity)*, kjer stranka personaliziral aplikacijo, določene nastavitve pa si bodo aktivnosti tudi delile *(login data - Shared preferences)*,
- novičarsko stran *(Brest Info Feed Activity)*, kamor bo podjetje stranki dostavilo odziv na naročilo, predstavilo trenutne akcijske ponudbe ali druge informacije, nudilo poprodajne storitve idr.

Na shemi so skupno označene kot element *Activity*, saj se vse aktivnosti



Slika 4.5: Shema aktivnosti aplikacije.

obnašajo zelo podobno. Izjema je le *Main Activity*, ki pa le proži izbrano aktivnost.

4.2.4 Splošne nastavitve

S splošnimi nastavitvami mobilne aplikacije upravlja razred *SharedPreferences*. Ta razred ponuja splošen okvir, ki omogoča shranjevanje in nalaganje parov ključ-vrednost primitivnih podatkovnih tipov. Na izbiro je katerikoli primitivni podatkovni tip: *boolean, float, int, long, string*. Ti podatki se samodejno shranijo in obnovijo na vsaki seji tudi ko stranka aplikacijo zaključi in ponovno zažene.

Podatke ključ-vrednost, do katerih ima aplikacija dostop od koderkoli, lahko dojemamo kot nekakšne globalne spremenljivke mobilne aplikacije in so aplikaciji vedno dostopni. V razviti mobilni aplikaciji vsebujejo podatke o strankinem računu, dolžino intervala za osveževanje podatkov s strežnika ter opcijo zvočnih in vibracijskih signalov. Tipično te podatke nastavi stranka.

4.2.5 Stanje aplikacije

Tudi ko aplikacija navidezno teče, se operacijski sistem lahko odloči, da zaključi eno ali več aktivnosti. To pomeni, da je stanje aplikacije potrebno na nek način ohraniti. Splošno znani statični koncept tu ne deluje, saj se inicializirani viri ob zaključku aktivnosti sprostijo, nad čimer programer nima vedno nadzora. Android za ohranjanje kompleksnih podatkovnih struktur ponuja štiri možnosti:

- nastavitve aplikacije *(ang. Application Preferences)*,
- datoteke *(ang. Files)*,
- ponudnike vsebine *(ang. contentProviders)*,
- in podatkovno zbirko SQLite *(ang. SQLite Database)*.

Za mobilno aplikacijo je bila izbrana rešitev shranjevanja stanja s pomočjo podatkovne zbirke SQLite. Vsaka aktivnost ob zagonu vedno inicializira lasten primerek podatkovnega objekta, ki deduje razred *DataObject*. To je

abstrakten razred, ki vsebuje najpomembnejše metode in spremenljivke za delovanje podatkovnega objekta in definira nekatere abstraktne metode, katerih implementacija je prilagojena potrebam trenutne aktivnosti. Primer implementirane metode je metoda, ki skrbi za povezovanje objekta s podatkovno zbirko. Objekt otrok poleg lastnosti in metod starša lahko vsebuje tudi lastnosti ali metode, prilagojen potrebam svoje aktivnosti in implementira nekatere abstraktne metode starševskega objekta *DataObject*.

Atributi, potrebni za delovanje podatkovnega objekta *DataObject*, so prikazani v kodnem izseku 4.1.

Kodni izsek 4.1: Osnovni atributi abstraktnega razreda DataObject

```
protected int id;
// Določitev tipa podatkovnega objekta
protected int dataType;
// Stanje, ki se nanaša na to instanco podatkovnega objekta
// Primer: podatki kompletni/nekompletni, poslani/ niso poslani
protected int state;
// Tabela v SQLite, ki ji podatki pripadajo
protected String dbTableName;
// Kontekst, na katerega se nanaša podatkovni objekt
protected Context context;
// Upravljalec podatkovne zbirke dodeljen podatkovnemu objektu
protected DatabaseHelper dbHandler;
// Podatkovna zbirka, v kateri se nahaja tabela dbTableName
protected SQLiteDatabase database;
```

Privzeti konstruktor v kodnem izseku 4.2 najprej določi, na kateri kontekst se trenutni podatkovni objekt nanaša in za kateri tip gre. Poleg tega si izdela t.i. pomagalnik za uporabo podatkovne zbirke *DatabaseHelper*, ki se nanaša na isti kontekst. S pomočjo pomagalnika *DatabaseHelper* se podatkovni objekt poveže s podatkovno zbirko, s katero lahko nato komunicira. Podatkovno zbirko mora pri vsakem dostopu odpreti in po koncu zapreti zato je priporočljivo, da se pri enem dostopu izda čimveč potrebnih zahtevkov in

tako omeji število dostopov do podatkovne zbirke.

Kodni izsek 4.2: Konstruktor razreda DataObject.

```
public DataObject(Context context, int dt){
  this.setDataType(dt);
  this.state=STATE_NOT_IN_DB;
  this.context=context;

  dbHandler = new DatabaseHelper(context, null);
}
```

Do atributov objekta je mogoč dostop le preko ustreznih *get* in *set* funkcij kot na primer do atributa *state* v kodnem izseku 4.3.

Kodni izsek 4.3: Dostop do atributa state.

```
public int getState() {
  return state;
}

public void setState(int s) {
  this.state = s;
}
```

DataObject definira tudi nekatere abstraktne metoda, ki jih implementira vsak dedujoči objekt, da je mogoče operirati s podatki podatkovne zbirke. Najbolj tipični metodi sta *load(int id)* in *store(int id)*, s pomočjo katerih vrstico iz ustrezne tabele naložimo oziroma jo shranimo. Primer je prikazan v kodnem izseku 4.4.

Kodni izsek 4.4: Abstraktne metode razreda DataObject

```
// naloži DataObject iz podatkovne zbirke
public abstract void load(int id);

//shrani DataObject v podatkovno zbirko
public abstract void store(int id);
```

```
//izbriši DataObject iz podatkovne zbirke
public abstract void delete(int id);
```

Tako je za vprašalnik o namembnosti opreme prostora izdelan podatkovni objekt *QuestionnaireObject* ki deduje razred *DataObject* implementira metodo *store(int id)*, ki se uporablja za shranjevanje informacij v podatkovno zbirko, pridobljenih od stranke, kot je prikazano v kodnem izseku 4.5. Navodilo *@Override* opozori prevajalnik, da gre za dedovano metodo.

Kodni izsek 4.5: Implementirana metoda store(int id) v razredu QuestionnaireObject.

```
public class QuestionnaireObject extends DataObject {
  ...

  @Override
  public void store(int id) {
    // sklad vrednosti, ki bodo shranjene v podatkovno zbirko
    ContentValues insertValues = new ContentValues();

    // podajanje vrednosti podatkovnega objekta na sklad
    insertValues.put(KEY_COMPLETED, this.completed);
    ...

    // osveževanje vrednosti vrstice tabele
    database.update(TABLE_QUESTIONNAIRE, insertValues,
      KEY_ID + " = ?", new String[] { String.valueOf(id) });
  }
}
```

Omenjeni podatkovni objekt *QuestionnaireObject*, ki deduje objekt *DataObject*, torej vsebuje podatke, ki se nanašajo na naročilo stranke. Na enak način je bil razvit tudi objekt *MessageObject*, ki vsebuje podatke o sporočilu, ki ga stranki pošlje podjetje kot odgovor na naročilo. Načrtovan je tudi po-

datkovni objekt *NewsObject*, ki bo vseboval informacije o novicah podjetja.

4.2.6 Vmesnik podatkovne zbirke

Razred *DatabaseHelper* nastopa kot vmesnik med podatkovnim objektom in podatkovno zbirko SQLite. Deduje razred *SQLiteOpenHelper*, katerega metode je potrebno implementirati po potrebah mobilne aplikacije. Uporablja se za odpiranje / zapiranje podatkovne zbirke (npr. metoda *getReadableDatabase()* in za upravljanje različnih verzij podatkovne zbirke (metoda *onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)*).

4.2.7 Razred REST

Objekt, ki skrbi za komunikacijo med aktivnostjo in strežnikom. Sledi REST arhitekturi. Za pošiljanje/sprejemanje podatkov uporablja podatkovni objekt aktivnosti oziroma storitve mobilne aplikacije v ozadju.

Po inicializaciji REST poskrbi za izdelavo pravilnega zahtevka REST. Sestavljen je iz:

- URIja ki vsebuje naslov domene, številko vhoda, prikazno ime, vzorec URL, začetno pot spletne storitve in ime vira,
- eno izmed metod HTTP post, get, delete ali put,
- in podatkov ki so lahko tipa slika, objekt JSON, string.

Primer gradnje lokacije vira je podan v kodnem izseku 4.6.

Kodni izsek 4.6: Metoda getURL ki vrne lokacijo vira.

```
private String getURI(String rc) {
  return "http://"+DOMAIN+":"+PORT+"/"
    +DISPLAY\_NAME+"/"+URL\_PATTERN+"/"+TOP_PATH+"/"+rc;
}
```

Kodni izsek 4.7 prikazuje, kako objekt REST pošlje izpolnjen vprašalnik stranke strežniku. V vsaki funkciji so vsi koraki, ki jih je potrebno izvesti, da se pravilno zgradi klic po priporočilih arhitekturnega načina REST ki

so pridobitev URIja, določitev metode HTTP, napoved tipa podatka, ki ga zahtevek vsebuje in vključitev podatka ustreznega tipa.

Kodni izsek 4.7: Funkcija ki strežniku pošlje izpolnjen vprašalnik v obliki objekta JSON.

```
public void sendOrder(final QuestionnaireObject qObject)
  throws ClientProtocolException, IOException {

  // Nova nit, ki je lastna samo temu zahtevku
  Runnable runnable = new Runnable() {
    private HttpResponse response;

    @Override
    public void run() {
      // ime končnega vira na strežnikovi strani
      rest_call="postOrderRequest";

      // inicializacija HTTP odjemalca
      HttpClient client = new DefaultHttpClient();
      // uporaba metode POST
      HttpPost post = new
          HttpPost(getURL(rest_call))resourceLocation;
      // opis vsebine, v tem primeru je to JSON objekt
      post.addHeader("Content-Type", "application/json");
      // pridobitev telesa klica REST
      try {
        post.setEntity(new StringEntity(
          (getJSON(qObject)).toString()));
      } catch (UnsupportedEncodingException e1) {
        e1.printStackTrace();
      }
      // izstavitev klica REST
      try {
        this.response = client.execute(post);
```

```
      } catch (Exception e) {
        System.out.println("response exception "+e);
      }
    }
  };
  new Thread(runnable).start();
}
```

Spletna storitev streže klicem objekta REST. Podrobneje je opisana v poglavju 4.2.11. Mobilna aplikacija se delovanja spletne storitve ne zaveda.

4.2.8 Osnovni tok dogodkov aktivnosti mobilne aplikacije

Ukaz stranke: Stranka izbere željeno aktivnost v glavnem meniju, na primer naročilo pohištva za prostor. Aplikacija odpre ustrezen vprašalnik in zbere informacije za arhitekta.
Reakcija aplikacije: Zažene se nova aktivnost. Oštevilčenje se nanaša na sliko 4.5.

1. Aktivnost kreira nov podatkovni objekt *DataObject*, ki vsebuje vse parametre, ki jih aplikacija pridobi od stranke.
2. Podatkovni objekt želi preveriti, če obstajajo že izpolnjeni podatki delno izpolnjenega naročila. Podatke dobi preko objekta *DatabaseHelper*.
3. *DatabaseHelper* se poveže na lokalno podatkovno zbirko in izda ustrezno zahtevo.
4. Podatkovna zbirka vrne zahtevane podatke, če ti obstajajo.
5. Podatki se zapišejo v podatkovni objekt.
6. Polja in gumbi aktivnosti se ustrezno nastavijo, stanje morebitnih prejšnjih izbir je obnovljeno.
   (a) Stranka lahko sproži oddajo naročila.
   (b) Izvrši se zahtevek REST in dostavi zbrane podatke spletni storitvi.

(c) Strežnik se odzove s potrditvijo.

(d) Aplikacija prikaže sporočilo o uspehu transakcije.

Stanje vprašalnika se sproti shranjuje v lokalno podatkovno zbirko. V primeru, ko stranka ne želi oddati naročila, se ob ponovnem zagonu aplikacije stanje vprašalnika le obnovi in stranka lahko nadaljuje postopek naročanja. Sistem sprotnega shranjevanja stanja vprašalnika stranki ponudi uporabniku prijazno izkušnjo, saj stranki ob ponovnem zagonu aplikacije ni potrebno ponovno izpolnjevati že izvedenih korakov vprašalnika.

4.2.9 Storitev mobilne aplikacije v ozadju

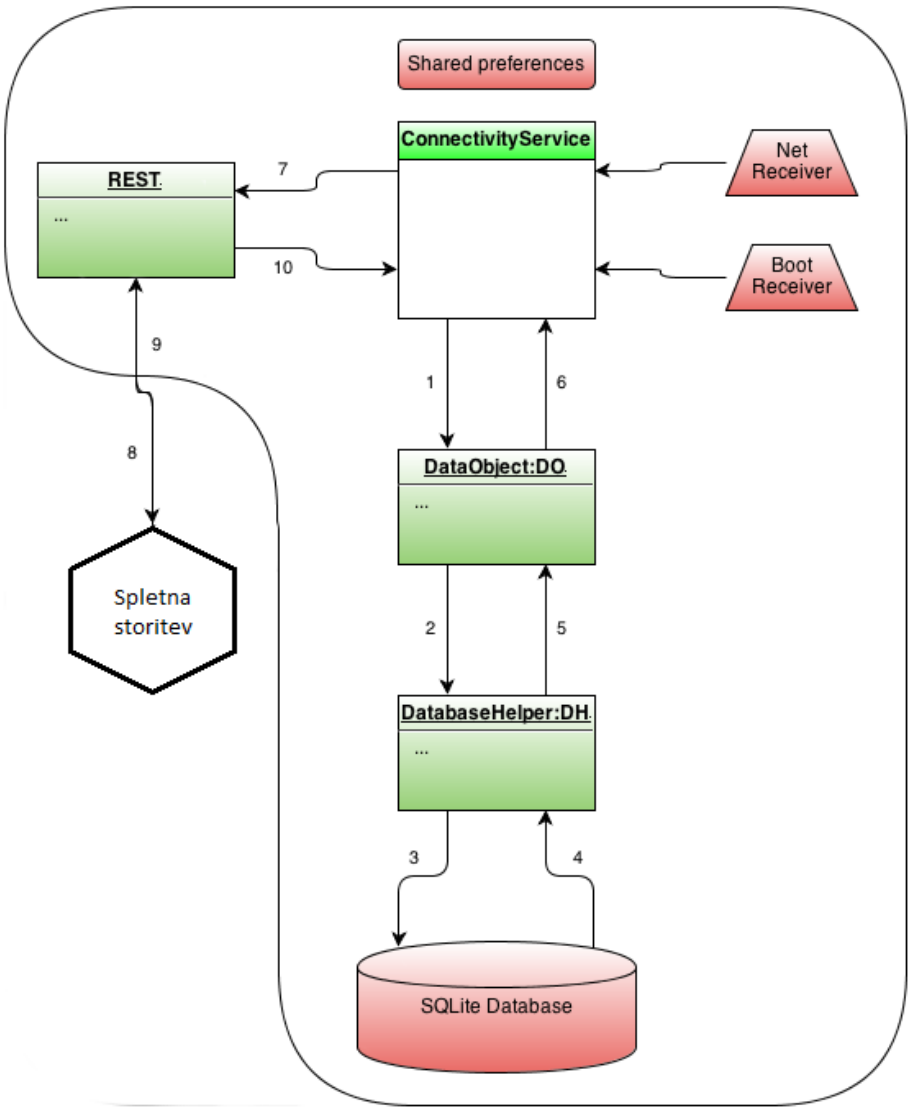
Bistvo storitve mobilne aplikacije v ozadju je, da z njo stranka ne rokuje neposredno in da skrbi za sinhronizacijo podatkov med aplikacijo in strežnikom. Storitev zažene operacijski sistem ob zagonu oziroma glede na razpoložljivost dostopa do interneta in ne stranka z zagonom aplikacije. Slika 4.6 prikazuje zgradbo storitve. Elementi, ki so obkroženi z okvirjem, sodijo v kontekst storitve.

Opozorilo: Storitev mobilne aplikacije (večkrat tudi storitev mobilne aplikacije v ozadju - kjer je poudarjen tek storitve tudi ob nedelovanju mobilne aplikacije) ni enaka spletni storitvi, saj prva teče na operacijskem sistemu Android in je del mobilne aplikacije, slednja pa teče na strežniku podjetja.

Temeljno delovanje storitve mobilne aplikacije v ozadju je enako aktivnosti. V nadaljevanju so opisani le objekti, ki so specifični za delovanje storitve. Elementi *Shared Preferences, DataObject, DatabaseHelper, SQLite, REST* in *spletna storitev* so opisani v podpoglavju 4.2.2, saj pri storitvi mobilne apilkacije nosijo enako vlogo.

4.2.10 Povezovalna storitev

Storitev *ConnectivityService*, ki teče v ozadju, skrbi za povezljivost s strežnikom podjetja. Kodni izsek 4.8 prikazuje ukaz za zagon storitve. Implementirati je potrebno vsaj funkcije *onCreate()*, *onStartCommand(Intent intent,*



Slika 4.6: Shema storitve mobilne aplikacije, ki teče v ozadju.

int flags, int startId) ter *onDestroy()* in funkcijo *run()*, ki jo uporablja nit.

Kodni izsek 4.8: Zagon storitve mobilne aplikacije v ozadju.

```
// zagon storitve
Intent i= new Intent(context, ConnectivityService.class);
context.startService(i);
```

Preko podatkovnih objektov storitev mobilne aplikacije v ozadju polni lokalno podatkovno zbirko SQLite z odzivom podjetja na naročilo, novimi akcijskimi ponudbami, poprodajnimi storitvami in ostalimi informacijami. Hkrati je potrebno paziti na čimmanj dostopov storitve do strežnika zaradi zmanjšanja porabe energije z ustrezno nastavitvijo dolžine posodobitvenih intervalov.

Storitev mobilne aplikacije teče v lastni niti. Ciklično preverja, ali je potrebno na strežnik prenesti podatke ali pa jih z njega pridobiti. Med cikli čaka določen čas, ki je reda nekaj ur. Za vsak tip podatkovnega objekta izda po en zahtevek tipa REST. Naročila storitve opremljanja prostora ne pošlje storitev mobilne aplikacije, zahtevek REST pošlje stranka iz aktivnosti (tega se stranka ne zaveda). Implementacija preproste niti je prikazana v kodnem izseku 4.9.

Kodni izsek 4.9: Implementacija niti s pomočjo katere storitev mobilne aplikacije osvežuje podatke.

```
@Override
public void run() { //
  // deklariramo not podatkovni objekt
  // ki ga želimo prejeti s strežnika
  MessageObject mo = new MessageObject(connectivityService,
      MessageObject.DATA_RESPONSE);
  // zahtevek bo tipa REST
  REST rest;

  // zahtevke izdajamo, dokler storitev teče
  while (ConnectivityService.this.runFlag) {
```

```
    // določi se končno lokacijo vira na strežniku (del URIja)
    rest_call="message";
    // izda se zahtevek
    rest = new REST(getBaseContext(), mo);
    // sporočilo se zapiše v podatkovni objekt mo
    rest.getMessage(mo);
    // do novega obhoda zanke se čaka DELAY časa
    Thread.sleep(DELAY);
  }
}
```

Poganjanje storitve v ozadju pomeni določeno porabo električne enenrgije. Poraba se zmanjša z določitvijo daljšega časovnega intervala in s pomočjo signalov operacijskega sistema o povezanosti z medmrežjem. Sprejemnik *Net receiver* prejema obvestila operacijskega sistema o povezljivosti. Ko povezava na internet ne deluje se storitev mobilne aplikacije zaustavi, ko pa se povezava vzpostavi, pa se storitev znova zažene. Namen tega sprejemnika je, da zmanjša porabo energije tako, da ustavi delovanje storitve mobilne aplikacije takrat, ko mobilna naprava nima dostopa do interneta.

Podobno deluje tudi sprejemnik *Boot receiver*, ki prejme obvestilo operacijskega sistema o zagonu naprave. Skrbi, da storitev mobilne aplikacije začne teči ob zagonu naprave (telefona/tablice), vendar ne vpilva na porabo energije ugodno.

4.2.11 Osnovni tok dogodkov storitve mobilne aplikacije

S pojmom osvežitveni cikel se označuje metode, ki se izvršijo takrat, ko storitev mobilne aplikacije v ozadju zahteva sveže podatke s strežnika podjetja oziroma pošlje generirane podatke uporabnika strežniku podjetja. Izvršuje se lahko več osveževalnih ciklov ki med sabo niso povezani oziroma odvisni. Po poteku nekega časovnega intervala se sproži eden izmed osvežitvenih ciklov. Neobičajni poteki dogodkov niso realizirani, razvoj je osredotočen le na ide-

alnega. Cikli se razlikujejo samo po vrsti podatkovnega objekta in zahtevku REST, zato jih na sliki 4.6 obravnavamo enako.

Opomba: števila pred vsakim korakom niso zaporedna, ampak se nanašajo na številske označbe sheme na sliki 4.6.

1. Osvežitveni cikel kreira podvrsto podatkovnega objekta *DataObject*, ki ustreza namenu cikla.

Nato imamo dve možnosti, pošiljanje podatka iz lokalne podatkovne baze strežniku in prejemanje podatkov strežnika in pisanje v lokalno podatkovno bazo.

Prva možnost: podatkovna zbirka SQLite → spletna storitev na strežniku:

2. Podatkovni objekt *DataObject* želi pridobiti izpolnjene podatke iz lokalne podatkovne zbirke. To stori s pomočjo vmesnika podatkovne zbirke *DatabaseHelper*.
3. *DatabaseHelper* se poveže na lokalno podatkovno zbirko in izda ustrezno zahtevo SQL.
4. Podatkovna zbirka *SQLiteDatabase* vrne zahtevane podatke, če ti obstajajo.
5. Podatki se zapišejo v podatkovni objekt.
6. Osvežitveni cikel storitve mobilne aplikacije *ConnectivityService* lahko prične s pošiljanjem podatkov.
7. Osvežitveni cikel zahteva ustrezno akcijo s pomočjo objekta REST.
8. Objekt REST generira in izda ustrezen zahtevek REST.
9. Spletna storitev s strežnika se odzove.
10. Objekt REST posreduje odziv osveževalnemu ciklu.

Druga možnost: spletna storitev s strežnika → podatkovna zbirka SQLite:

7. Osvežitveni cikel storitve mobilne aplikacije *ConnectivityService* zahteva ustrezno akcijo s pomočjo objekta REST (npr. pridobi odziv arhitekta).
8. REST objekt generira in izda ustrezen zahtevek REST (v tem primeru običajno s HTTP metodo GET).

9. Spletna storitev s strežnika se odzove s paketom podatkov (vsebujejo odziv arhitekta, slike opremljenega prostora itd...).
10. Objekt REST posreduje odziv osvežitvenemu ciklu.

1.a Osvežitveni cikel storitve mobilne aplikacije pridobljene podatke posreduje svojemu podatkovnemu objektu *DataObject*.

2. Podatkovni objekt želi posredovati pridobljene podatke lokalni podatkovni zbirki *SQLite Database*. To stori s pomočjo *DatabaseHelperja*.
3. *DatabaseHelper* se poveže na lokalno podatkovno zbirko *SQLite Database* in izda ustrezno zahtevo SQL.
4. Podatkovna zbirka *SQLite Database* shrani posredovane podatke in izda sporočilo o uspešnem zapisovanju.
5. Sporočilo o uspešnem zapisovanju.
6. Sporočilo o uspešnem zapisovanju. Cikel se zaključi. Podatki so pripravljeni za prikaz takoj, ko stranka požene ustrezno aktivnost.

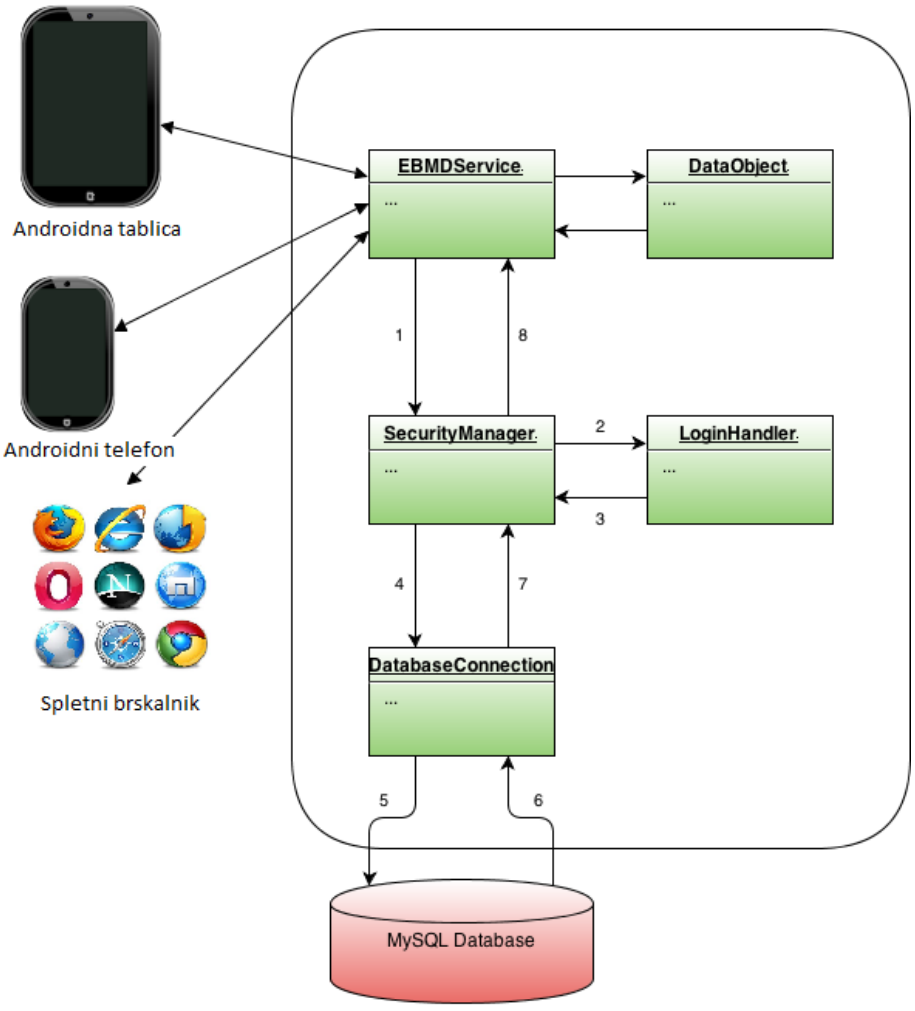
V primeru, ko storitev mobilne aplikacije od spletne storitve pridobi sporočilo, ki je namenjeno stranki osebno (npr. odziv arhitekta na naročilo, poprodajne storitve...), se pojavi opozorilna ikona v zgornji opozorilni vrstici zaslona, mobilna naprava pa se odzove z zvočnim signalom, če je stranka izbrala to možnost v splošnih nastavitvah.

4.3 Spletna storitev

Spletna storitev je namenjena servisiranju HTTP zahtev z arhitekturo REST, ki običajno prihajajo z naprav Android. Storitev teče na spletnem strežniku Apache Tomcat. Za upravljanje s podatkovnimi zbirkami je uporabljen MySQL. Elementi na sliki 4.7, ki so obkroženi z okvirjem, sodijo v kontekst spletne storitve.

4.3.1 Arhitektura spletne storitve

Arhitektura spletne storitve je bila izdelana za arhitekturo mobilne aplikacije in zaradi poenostavitve sledi le tej, kjer je to smiselno. Predvsem pri podat-



Slika 4.7: Shema spletne storitve ki streže zahtevkom, ki pridejo do strežnika.

kovnih objektih, ki hranijo enake informacije na obeh sistemih, se je enaka zasnova izkazala za pravo izbiro. Primer je lahko razred *MessageObject*, ki vsebuje informacije o sporočilu. Ta objekt napolni oziroma izdela zaposleni na podjetju, nato je pripravljen, da ga prejme odjemalec - mobilna aplikacija in ga prikaže stranki. Slika 4.7 prikazuje arhitekturo spletnege storitve. Podatkovni objekt *DataObject* deluje enako kot pri mobilni aplikaciji, zato tu ni posebej opisan. S spletno storitvijo komunicirajo prenosne naprave Android, ki delujejo kot odjemalci. Na njih teče mobilna aplikacija EkoBuMDesign.

Spletna storitev lahko streže tudi klicem z brskalnika, ki so namenjeni predvsem upravljanju s podatki s strani podjetja, kot je na primer izdelava novic ali akcijske ponudbe, izdelava odgovora stranki ipd. V okviru diplomske naloge ta del ni realiziran oziroma je vključen samo preprost klic s preprostim odzivom in služi kot nastavek za nadaljni razvoj. Podatkovna zbirka MySQL teče na strežniku. Podrobneje je opisana v podpoglavju 4.3.1.

Vstopna točka spletne storitve se imenuje *EBMDService*, kjer se opravi detekcija, za katero vrsto klica gre in kakšen tip podatkov je poslan oziroma zahtevan. Običajno je pri dostopu do vira strežnika določena lokacija, metoda HTTP in vrsta vsebine dostopa. Kodni izsek 4.10 podaja način, kako lahko mobilna aplikacija strežniku dostavi fotografijo. Zahtevek REST določi lokacijo vira, ki je na relativni lokaciji *.../WebService/postOrderRequestPictures*, metoda HTTP, ki je v danem primeru *POST* in vsebino, ki je tipa *MediaType.MULTIPART_FORM_DATA* oziroma gre za slikovno predstavitev. V spletni storitvi je običajno določena tudi vrsta odgovora *(ang. response)*, ki se lahko med drugim uporabi za sporočanje uspešnosti prejema zahtevka. V podanem primeru je odgovor formata *text/plain*, ki predstavlja navaden poljuben niz.

Kodni izsek 4.10: Primer dela zgradbe spletne storitve.

```
@Path("/WebService")
@Singleton
public class EBMDService {
  // implementacija krovnega razreda
```

```
  @POST
  @Path("/postOrderRequest")
  @Produces("text/plain")
  @Consumes("application/json")
  public String postOrderRequest(String json) throws Exception {
    // implementacija prejema naročila
  }

  @POST
  @Path("/postOrderRequestPictures")
  @Produces("text/plain")
  @Consumes(MediaType.MULTIPART_FORM_DATA)
  public String postOrderRequestPictures(@Context
      HttpServletRequest request) {
    // implementacija prejema fotografije
  }
}
```

Upravljalec varnosti imenovan *SecurityManager* določi, za kakšno vrsto klica gre in po potrebi (gre za pošiljanje/sprejemanje osebnih podatkov ali samo za prejemanje novic podjetja?) zahteva preverbo identitete s pomočjo razreda *LoginHandler*. Ko je istovetnost uzdajatelja potrjena, se dovoli dostop do podatkovne zbirke s pomočjo razreda *DatabaseConnection*.

Osnovni tok dogodkov spletne storitve

Osnovni tok dogodkov je ilustriran na sliki 4.7.

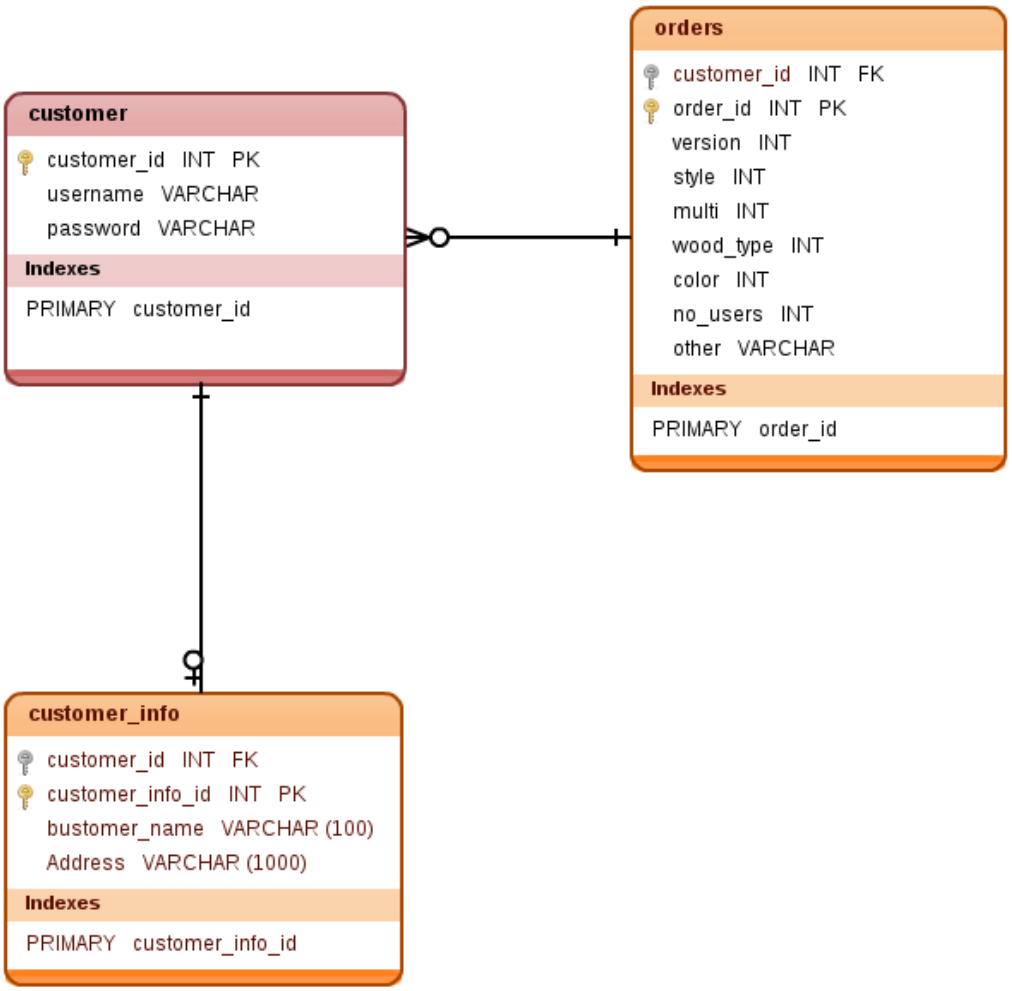
1. Spletna storitev se odzove na zahtevek REST s klicem razreda *SecurityManager*.
2. *SecurityManager* po potrebi zahteva preverbo identitete.
3. *LoginManager* potrdi identiteto.
4. *SecurityManager* poda zahtevo za dostop do podatkovne zbirke.

5. *DatabaseConnection* se poveže na podatkovno zbirko, izdela in izda ustrezen stavek SQL.
6. Podatkovna zbirka pri pisanju izda sporočilo o uspehu, pri branju pa dostavi podatke.
7. *DatabaseConnection* preda sporočilo/podatke objektu *SecurityManager*.
8. *SecurityManager* preda podatke spletni storitvi.
9. Spletna storitev se ustrezno odzove na podani zahtevek.

4.4 Podatkovna zbirka

V podatkovni zbirki hranimo preprosto tabelo naročil, ki jih prejme spletna storitev. Ker ta del rešitve celotnega problema ni tema tega diplomskega dela, služi izdelana podatkovna zbirka samo kot nastavek za kasnejši razvoj.

Uporabljena podatkovna zbirka se imenuje *brest*. Vsebuje tri tabele. Tabela *customer* vsebuje osnovne podatke o strankah, tabela *customer_info* podatke, ki jih podjetje potrebuje za dostavo produktov in izvedbo finančnih transakcij in tabela *orders* ki vsebuje podatke o naročilih storitve podjetja. Slika 4.8 prikazuje strukturo podatkovne zbirke sistema.



Slika 4.8: Struktura podatkovne zbirke

Poglavje 5

Testiranje

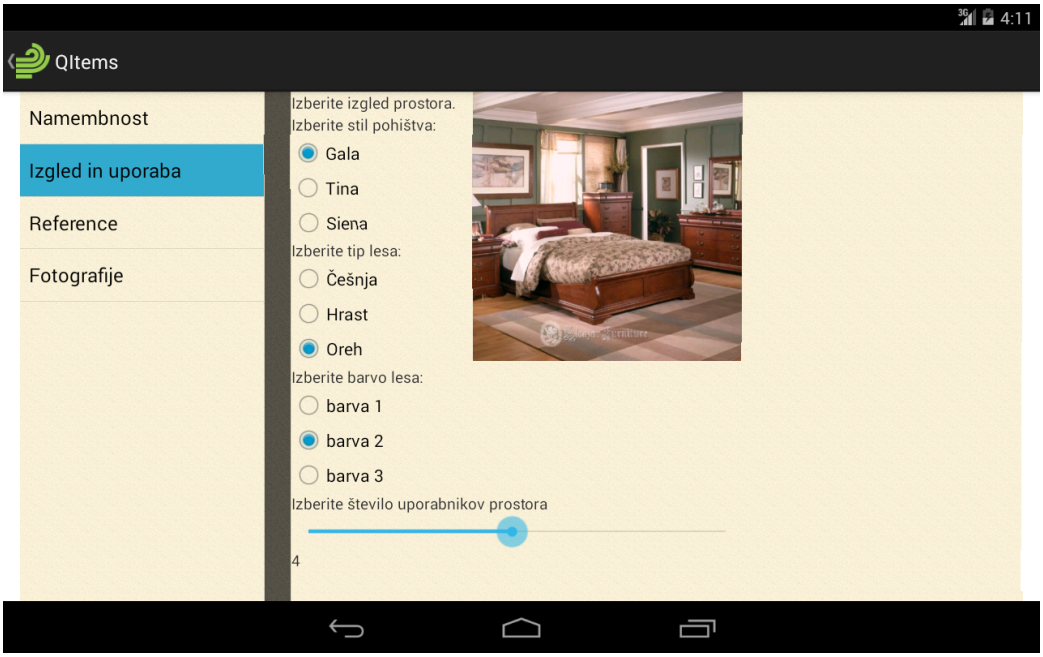
Mobilno aplikacijo smo preizkusili na različnih mobilnih napravah. Predvsem smo upoštevali različne velikosti zaslona in pa različne verzije operacijskega sistema. Pogoji za delovanje vseh funkcionalnosti so operacijski sistem verzija 3.0 ali novejši, v napravo pa je vgrajen pospeškometer. Slednja funkcionalnost je potrebna le pri prikazu interaktivnega prostora in ni nujno potrebna pri izvedbi naročila.

Tekom razvoja prototipa smo napisali okrog 6000 vrstic kode, kar je bilo ugotovljeno s pomočjo vključka za razvojno okolje Eclipse za analizo programske kode Metrics. Vse potrebne datoteke za preizkus sistema se nahajajo na spletni strani laboratorija LUSY [3].

5.1 Testirani sistemi

Aplikacijo EkoBuMDesign smo testirali na naslednjih napravah:

Asus Nexus 7 (2012), Android 4.2.2 in Android 4.4.4
Aplikacija deluje po pričakovanjih, saj je bila razvita ravno na tej napravi. V razvojnem okolju je bila postavitev zaslonskih elementov prilagojena velikosti zaslona te naprave (Slika 5.1). Ostale funkcionalnosti kot so prikazovanje strojno pospešene grafike, povezljivost, procesi v ozadju, delujejo ravno tako



Slika 5.1: Pogled testnega zaslona na napravi ASUS Nexus 7 2012.

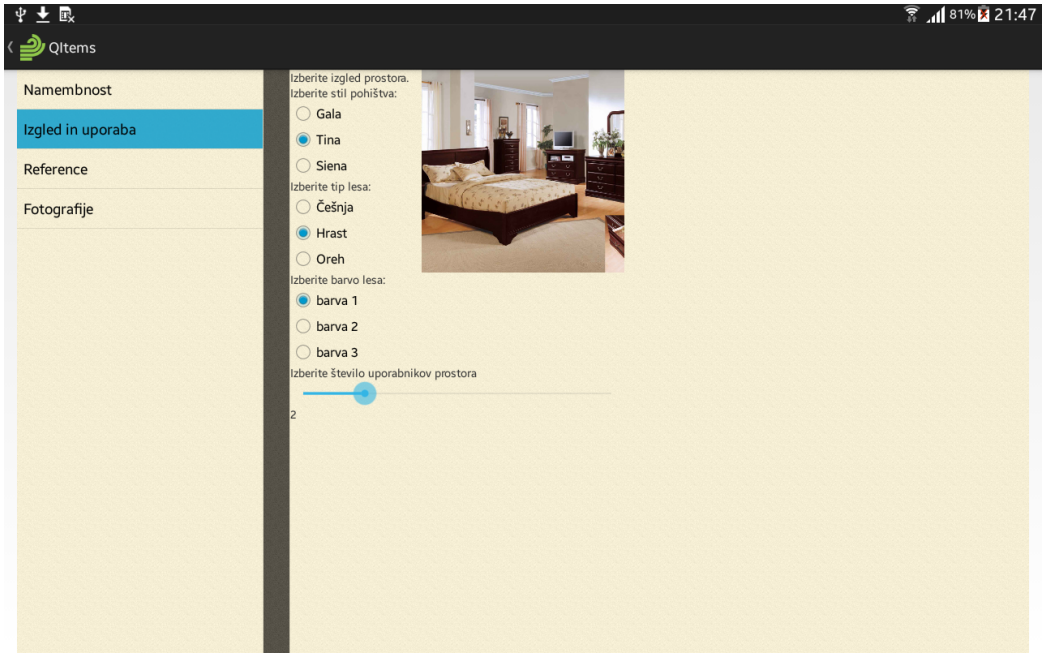
brez težav.

Samsung Galaxy Tab 4, Android 4.4.2

Aplikacija deluje normalno, na zaslonu pa so elementi opazno večji kot pri prototipni napravi, saj ima zaslon Galaxy Taba večjo površino pri enaki gostoti pik (Slika 5.2). Posebnosti pri delovanju ni opaziti.

Sony Xperia M, Android 4.3

Na napravi aplikacija deluje, vendar je prikazna površina manjša kot pri prototipni napravi. Tu v veljavo stopijo prilagoditve za manjše zaslone, ki smo jih upoštevali pri razvoju aplikacije za več velikosti prikaznih površin. Pri aktivnosti, kjer je na voljo več zavihkov, so pri tej napravi zavihki prikazani čez celoten zaslon, vsaka izbira zavihka pa popolnoma prekrije vse zavihke. Na tak način smo omogočili dobro uporabniško izkušnjo tudi na manjši površini (Slika 5.3, 5.4), vendar bi razporeditev elementov na zaslonu potrebovala



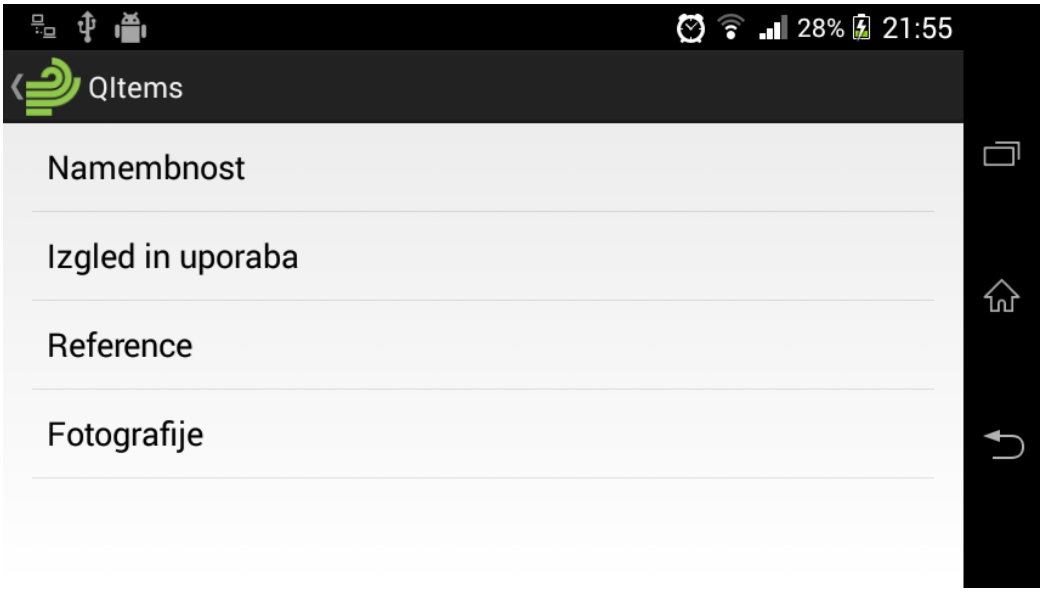
Slika 5.2: Pogled testnega zaslona na napravi SAMSUNG Galaxy Tab 4.

še dodatne prilagoditve. Ostale funkcionalnosti, ki so uporabnikovim očem skrite, delujejo normalno.

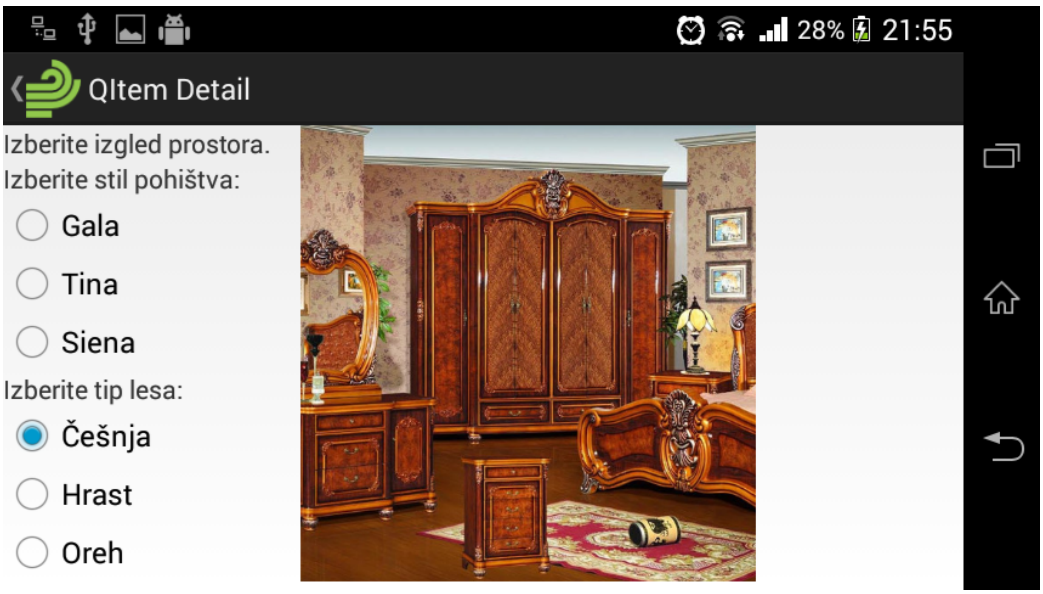
5.2 Poraba sistemskih virov

Aplikacija v pomnilniku naprave zaseda relativno veliko prostora, okrog 5,2 MB. Za zmanjšanje porabe prostora bi bila potrebna optimizacija predvsem slikovnega gradiva, vendar je v tej fazi izvedbe nismo predvideli.

Aplikacija je namenjena končnemu uporabniku, zato je od natančno izmerjenih odzivnih časov pomembnejša uporabniška izkušnja. Zagonski čas smo vseeno izmerili in sicer smo testiranje izvajali za vsako testno napravo posebej. Hitrost zagona je bila izmerjena od začetka izvajanja metode *onCreate()*, ki je vstopna točka aplikacije in do konca izvajanja metode *onPostResume()*, ki je zadnja metoda, ki se izvede po izrisu zaslonske slike ko smo prepričani, da so se izvedli vsi vmesni koraki življenjskega cikla aplikacije.



Slika 5.3: Pogled testnega zaslona na napravi SONY Xperia M. Prikazan je izbirni meni.



Slika 5.4: Pogled testnega zaslona na napravi SONY Xperia M. Prikazana je izbera elementa menija.

Časi so za vsako napravo precej različni, saj se naprave močno razlikujejo po količini pomnilnika, zmogljivosti procesorja in drugih strojnih komponentah. Rezultati meritev so prikazani v tabeli 5.1. Interaktivne izbire v sami aplikaciji se izvedejo takoj. Nalagalnih časov med uporabo aplikacije ni. To pomeni, da uporabnik ne doživlja frustracij med uporabo aplikacije.

Ime naprave	Povprečni zagonski čas
ASUS Nexus 7	212 ms
SAMSUNG Galaxy Tab 4	156 ms
SONY Xperia M	280 ms

Tabela 5.1: Tabela prikazuje povprečne zagonske čase aplikacije za posamezno napravo.

5.3 Spletna storitev

Spletna storitev je bila razvita v namene prototipiranja in ponuja le osnovne funkcionalnosti, zato ga nismo posebej testirali. Na zahteve katerekoli naprave se je odzival identično kar pomenii, da v tej fazi projekta deluje dovolj dobro. Pisanje v in branje iz podatkovne zbirke je ravno tako potekalo brez težav. Osnovni naslov spletne storitve je *http://brest.fri.uni-lj.si*, ki preusmeri zahtevo na naslov *http://193.2.76.45:8080*. Spletna storitev za sporočila je tako dostopna na naslovu *http://193.2.76.45:8080/si.brest.ekobumdesign.ws/REST/WebService/message*. Ta storitev aplikaciji pove, ali je na voljo sporočilo podjetja in če obstaja, ga aplikaciji tudi dostavi.

Poglavje 6

Sklepne ugotovitve in nadaljnje delo

V okviru diplomskega dela je bil razvit del rešitve problema podjetja: komunikacija med stranko in arhitektom, ki obsega spletno storitev in mobilno aplikacijo. Raziskane in uporabljene so bile ustrezne tehnologije pri realizaciji komunikacjskega kanala. Tako implementacija komunikacije sledi arhitekturnemu načinu REST, za strežnik in mobilno aplikacijo pa so uporabljene odprtokodne rešitve. Kot dobrodošla posledica odprtokodnosti je tudi znižanje stroškov in dobra podpora s strani odprtokodne skupnosti.

Prototip mobilne aplikacije realizira zahteve podjetja in hkrati ponuja tudi nekatere izboljšave, ki so se izkazale za koristne. Aplikacija najprej stranki omogoča naročilo storitve podjetja. S pomočjo implementiranega vprašalnika arhitekt od stranke pridobi vse informacije, da lahko ustrezno opremi strankin prostor. Aplikacija tudi vključuje možnost prejetja sporočila arhitekta o izdelani opremi prostora. Vključen je tudi prikaz ostalih informacij, ki obsegajo predvsem reklamna sporočila podjetja.

Strežniški del lahko prejema sporočila in jih vpisuje v podatkovno zbirko ter omogoča pošiljanje sporočil odjemalcem. Sporočila jemlje iz podatkovne zbirke na strežniku. Delo je bilo osredotočeno predvsem na izdelavo mobilne aplikacije, vendar je strežniški del za delovanje odjemalca nujno potreben,

tako da so bile razvite osnovne funkcionalnosti le tega.

Rezultat diplomskega dela izpolnjuje zastavljene cilje in jih po predstavitveni plati celo presega.

Ob zaključku izdelave diplomskega dela so se pokazale možnosti za izboljšanje projekta. Pri mobilni aplikaciji bi bilo mogoče izboljšati:

- uporabniški vmesnik,
- povezljivost s socialnimi omrežji,
- graditi uporabniško skupnost,
- dodati nove rubrike kot so nasveti za vzdrževanje izdelkov podjetja,
- uporabniku omogočiti interaktivno oblikovanje lastnega prostora v treh dimenzijah.

Na strani strežnika pa bi bilo mogoče izboljšati:

- povezljivost z obstoječimi sistemi podjetja,
- varnost,
- in izdelati uporabniški vmesnik za zaposlene.

Literatura

[1] Android architecture – the key concepts of android os. http://www.android-app-market.com/android-architecture.html. Nazadnje dostopano: september 2014.

[2] Delež verzij operacijskega sistema android novembra 2013. http://developer.android.com/about/dashboards/index.html. Nazadnje dostopano: januar 2014.

[3] Ekobumdesign - prodaja pohištva s pomočjo mobilnih tehnologij. http://lusy.fri.uni-lj.si/sl/node/110. Nazadnje dostopano: september 2014.

[4] Homestyler interior design. https://play.google.com/store/apps/details?id=com.autodesk.homestyler. Nazadnje dostopano: september 2014.

[5] Houzz interior design ideas. https://play.google.com/store/apps/details?id=com.houzz.app. Nazadnje dostopano: september 2014.

[6] Introduction to web services. http://www.w3schools.com/webservices/ws_intro.asp. Nazadnje dostopano: september 2014.

[7] Platform versions. http://developer.android.com/about/dashboards/index.html. Nazadnje dostopano: september 2014.

[8] Razširjenost mobilnih operacijskih sistemov v sloveniji med julijem 2008 in novembrom 2013. http://gs.statcounter.com/\#mobile_os-SI-monthly-200807-201311. Nazadnje dostopano: januar 2014.

[9] Uradna najava platforme android na googlovem blogu. http://googleblog.blogspot.com/2007/11/wheres-my-gphone.html. Nazadnje dostopano: september 2014.

[10] What is mysql? http://dev.mysql.com/doc/refman/4.1/en/what-is-mysql.html. Nazadnje dostopano: september 2014.

[11] Wikipedia o restu. http://en.wikipedia.org/wiki/Representational_state_transfer. Nazadnje dostopano: september 2014.

[12] Življenjski cikel aktivnosti. http://developer.android.com/about/dashboards/index.html. Nazadnje dostopano: september 2014.

[13] ISO/IEC 9075. *Information technology—Database languages—SQL*. International Organization for Standardization, Geneva, 2011.

[14] E. Burnette. *Eclipse IDE Pocket Guide*. O'Reilly, Sebastopol, 2005.

[15] C. Pautasso E. Wilde. *REST: From Research to Practice*. Springer, New York, 2011.

[16] T. Khare. *Apache Tomcat 7 Essentials*. Packt Publishing, Birmingham, 2012.

[17] R. Meier. *Professional Android Application Development*. Wiley Publishing, Indianapolis, 2009.

[18] J. Sandoval. *RESTful Java Web Services*. Packt Publishing, Birmingham, 2009.